Balke, Tina (Ed.); Yaich, Reda (Ed.)

Working Paper

Proceedings of the 12th European Agent Systems Summer School Student Session

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik, No. 50

Provided in Cooperation with:
University of Bayreuth, Chair of Information Systems Management

Suggested Citation: Balke, Tina (Ed.); Yaich, Reda (Ed.) (2010) : Proceedings of the 12th European Agent Systems Summer School Student Session, Bayreuther Arbeitspapiere zur Wirtschaftsinformatik, No. 50, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik, Bayreuth, https://nbn-resolving.de/urn:nbn:de:bvb:703-opus-7323

This Version is available at:
https://hdl.handle.net/10419/52633

Lehrstuhl für
Wirtschaftsinformatik
Information Systems
Management

No. 50
September 2010

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Tina Balke, Reda Yaich (Eds.)

Proceedings of the 12th European Agent Systems Summer School Student Session

Bayreuth Reports on Information Systems Management

UNIVERSITÄT
BAYREUTH

ISSN 1864-9300

Proceedings of the

12th European Agent Systems Summer School

Student Session

Preface

This volume contains the papers presented at the Student Session of the *12th European Agent Systems Summer School* (EASSS) held on 25th of August 2010 in Saint-Etienne, France.

The Student Session, organised by students, is designed to encourage student interaction and feedback from the tutors. By providing the students with a conference-like setup, both in the presentation and in the review process, students have the opportunity to prepare their own submission, go through the selection process and present their work to each other and their interests to their fellow students as well as internationally leading experts in the agent field, both from the theoretical and the practical sector.

As the goal of the Student Session is to provide the speakers with constructive feedback and a means to be introduced to the community, the competitive elements often found in conferences (best paper award, best presentation award) are intentionally omitted. Preparing a good scientific paper is a difficult task, practising it is the benefit of this session.

All submissions were peer-reviewed and accepted paper submissions are assigned a 30 minute slot for presentation and discussion at the Summer School. Typically a presentation either details the intended approach to a problem or asks a specific question, directed at the audience.

The review process itself was extremely selective and many good papers could not be accepted for the final presentation. Each submission was reviewed by at least 5 programme committee members, which decided to accept the 4 papers that are presented in these proceedings.

August 22nd, 2010

Tina Balke
Reda Yaich

Student Session Organization

Programme Chairs

Tina Balke
Reda Yaich

Local Organization

Reda Yaich

Programme Committee

Andrea Addis
Stéphane Airiau
Giulia Andrighetto
Haris Aziz
Nils Bulling
George Christelis
Irina Diana Coman
Fabiano Dalpiaz
Juergen Dix
Jim Duggan
Joseph El Gemayel
Torsten Eymann
Angela Fabregues
Berndt Farwer
José M. Gascueña
Nicola Gatti
Cristian Gratie
Carlos Grilo
Davide Grossi
Akın Günay
Martin Günther
Anthony Hepple
Hanno Hildmann
Benjamin Hirsch
Enda Howley
Sebastian Hudert
Franziska Klügl
Andrew Koster
Ramachandra Kota
Stefan König
Tobias Küster
Benoit Lacroix
João Leite
Shuangyan Liu
Brian Logan
Pablo Lucas
Marin Lujak
Angel Rolando Medellin
Christoph Niemann
Paulo Novais
Julian Padget
Mario Paolucci
Antonio Pereira
Isaac Pinyol
Michele Piunti
Eric Platon
Evangelos Pournaras
Abdur Rakib
Alessandro Ricci
Jordi Sabater Mir
Marco Schorlemmer
Julien Siebert
Robert Siegfried
Martin Slota
Jackeline Spinola de Freitas
Eugen Staab
Tomislav Stipancic
Leon van der Torre
Laurent Vercouter
Harko Verhagen
Serena Villata
Daniel Villatoro
Meritxell Vinyals
Danny Weyns
Yining Wu

Table of Contents

Distributed Reputation Extraction in Multi-Agent Systems

Andreea Urzica
Politehnica University of Bucharest
Splaiul Independentei 313
Bucharest, Romania
+33662599940
andreea.urzica@cs.pub.ro

ABSTRACT
Trust and reputation models are considered to be key issues in designing open multi-agent systems. They allow agents to synthesize the information needed for secure and efficient interactions when faced with uncertain situations. Existing literature shows many attempts of providing methods of evaluating trust and reputation but no consensus has been reached. Different and multiple components underlie the decision of trusting another agent. Various sources can be used for building one's reputation and assembling them all together. Our aim is to propose an approach that alleviates the agents from centralized reputation providers, by empowering them to extract reputation from data collections.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems, Intelligent agents.*

General Terms
Documentation, Performance, Design, Reliability

Keywords
Multi-Agent Systems, Trust, Reputation

1. INTRODUCTION
Virtual environments offer multiple opportunities for agent interactions: transactions on electronic markets, joining forces by using collaborative frameworks, information exchanges, etc. These interactions would not take place if the agent would not be able to estimate a certain degree of trust for an interaction partner. In order to compute the amount of trust towards a potential collaborator, an agent must collect and interpret information concerning his target as well as the context of the interaction.

The paper investigates what are the information sources that could aid an agent to decide whether entrust or not another in a transaction or a collaboration. After the interaction took place, the agent should be able to emit an opinion upon the unfolding of the interaction, the outcomes and his partner. By collecting the opinions of multiple agents on similar situations some conclusions can be drawn, conclusions that reflect the behavior of the actor involved in those situations. The evaluation resulted from aggregating multiple opinions is usually referred as the reputation of the given target. One's reputation constitutes a useful piece of information when we are faced with the decision of trusting him or not. The present paper shows how, by optimizing this circular influence between trust and reputation within a multiagent system, we can improve the system functionality regarding agent's benefit.

Cite as: Distributed Reputation Extraction in Multi-Agent Systems, Andreea Urzica, *Proc. of the 12th European Agent Systems Summer School (EASSS 2010)*, August, 23–27, 2010, Saint-Etienne, France.

An important feature we want to emphasize is the situational aspect of the evaluations employed by our model. Generally, reputation is computed and used as an absolute value, one-measure-fits-all kind of approach, using a single, designer-chosen aggregation model applied on all data that regards the target agent. We propose a more precise and personalized solution: precise, as the data collected is linked to the corresponding context, and personalized, as the queries are highly customizable, returning specific answers. A system design is provided, comprising a trust and a reputation model, both managed directly by the individual agent.

The paper proposes a system where agents are endowed with modules able to extract reputation information by independently aggregating opinions of other community members regarding situations similar to the one they face at the moment. The system provides one or more data repositories, containing raw, homogeneous, multidimensional data. The agent-side modules connect themselves to any data repository and select a subset of situations. The subset of situation corresponding to the query is then used to compute the value best reflecting the community opinion upon the given situation.

The paper is organized as follows. In Section 2, we motivate our approach by analyzing the current existing literature. Section 3 argues our vision upon building trust and the use of reputation. Section 4 investigates the information sources for building trust. Section 5 defines how the opinion concept is employed. Section 6 describes the repository component in the proposed model. Section 7 shows how reputation is extracted. Section 8 presents the axes for future work and Section 7 concludes.

2. RELATED WORK
In this paper, we are interested to see what the information sources that build trust are, how to extract the social reputation of the target, how is data aggregated and how is the information represented. An important amount of models (e.g. [2], [4], [5], [7], [18], [19]) build trust by relying on two main measures: the evaluator's own confidence, and the social reputation of the target.

Recently, more and more papers [22], [11], [6] agree that any trust or reputation value has a rather low significance in the absence of the facts arguing the decision. The reasons sustaining this view include the heterogeneity of agents within a system and the multiple possibilities of interpreting trust even when agents do utilize the same computational model. This usually arrives due to individual motivation or different perception capabilities (as they may have been developed by different companies and are pursuing individual goals). We consider that trust is context-dependent. The idea can be found in many works, e.g. [2], [16], [20], [13]. They state that "in real life, history of interaction will have to capture not only the outcomes, but also the context in which a certain outcome was produced" [20], and "information captured under certain conditions can become worthless under different conditions" [13]. In [3] for example, a paper proposing trust mechanisms for peer-to-peer networks, not only the resulted value, but also data about the observed behavior about agents is made directly available.

As shown in [15] and [11], the context can be communicated by defining situations. The model proposed in [11] has been of great inspiration for our work as it includes the formalization needed for representing situations. Situations are defined in [11] as tuples that can capture various aspects of a given interaction (e.g. actors' identity, timestamp, action, agent role, etc.).

The global view upon a target entity is more accurate and fine grained when the reputation values are associated to the situations involving the target and not to the target alone. The satisfaction values associated to the situation can be further aggregated in order to produce the reputation value of the situation. In [11] the tuples < situation, reputation value > are called "agreements" and they express the consensus reached in the space of reputation opinions sent by a set of agents about a particular situation. The model we propose is also based on tuples describing situations, the association between situations and agents' satisfaction value and association between a situation and the aggregation of the associated set of satisfaction values. In our view, the aggregated value does not represent a consensus among those who have expressed their opinions: firstly because they emitted the opinion solely to reflect their evaluation on the output of the interaction, and not with the purpose of finding a consensus and secondly as the role of the aggregation function is usually exactly the one of finding a compromise between divergent opinions.

We consider interesting the work found in [6] as it introduces the idea of a hybrid model, attempting to solve problems associated with both centralized and decentralized reputation models.

An inspiring trust model that defines an agent-based aggregation engine is presented in [22]. It computes the trustworthiness of candidates by aggregating their historical contractual evidences and taking into account important properties of the dynamics of trust by using a sin-based aggregation curve. We prospect the idea of using a similar curve in our model, one that should able to handle the multiple information sources employed by our trust computation module, some of essentially different nature (e.g. the set of situations recording the history of encounters and digitally signed certificates).

In approaches like [17] or [18] reputation consists of a universal value provided by a global module that applies a predefined aggregation function on the entire dataset. By contrast, our model the agent is able to select the situations he is interested in and perform any aggregation on the ratings associated to those situations only.

3. OVERVIEW OF THE TRUST MODEL

This section explains our vision upon how trust is built and how opinions should be used. It stresses the distinction between the subjective evaluation of the trustworthiness degree of a potential interaction partner, and the opinion issued by the same agent upon his collaborator, after the interaction, if the interaction took place.

Figure 1 describes the model organization, placing in a temporal framework the dependency between the operations and the concepts employed.

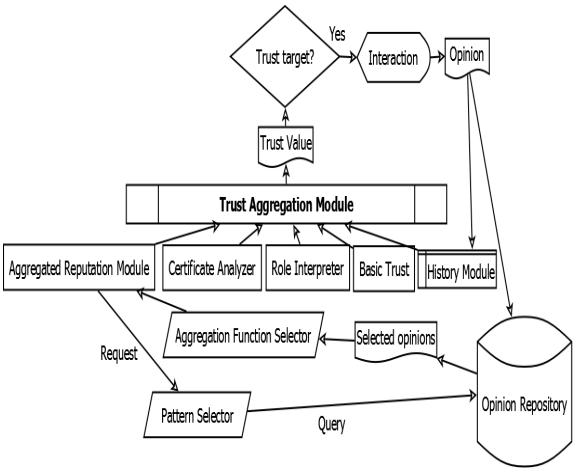


Figure 1 – System Architecture

By the term trust we refer the information computed and used when faced with the decision of whether accepting or not a contract and relying on a partner. The input information needed for computing the amount of trust toward another agent includes a rather wide range of factors: previous personal experience (history), information about roles, certificates signed by trusted authorities, etc.

The Trust Aggregation Module, depicted in Figure 1, uses trust information from various sources (shown in Section 4), in order to compute the level of trustworthiness. We propose that each information source should be handled by a module that stores and manages the records. The modules may also perform some data pre-processes, such as normalization or conversion, thus working as an adapter between the information source and the Trust Aggregation Module.

Depending on the amount of trust computed by the Trust Aggregation Module and an agent-specific threshold, the agent decides if an interaction can or cannot take place (as shown in Figure 1). The interactions may be of various types: synchronous or asynchronous, monetary or non-monetary transactions or

collaboration between team members, using or not using norms, etc. Specifying the type of interaction is out of the scope of this paper. The result of the entrusting decision may be of binary nature (the agent either interacts, either not) or, it may imply a set of adjustments brought to the contract clauses and a re-negotiation in order to ensure a better protection to risks.

After an interaction took place, an agent is able to express an opinion (see Section 5) as the degree of fulfillment of the quality variables agreed upon in the contract and of how much the output of the interaction met the evaluator's personal preferences. One copy of the resulted opinion is added to the agent's History Module to be further used as personal experience, while another copy is sent to the Opinion Repository in order to become available to the entire community.

We consider opinions to be an important source of raw material to be processed. While very specific and fine grained, they have the advantage of allowing selective aggregation. Queries can easily select the opinions by the fields of interest within the situation they concern. Thus, the opinions filed in an Opinion Repository can be combined in multiple, customizable ways, according to the requester's preferences and interests. This way a user can extract the information that answers his specific needs. There are users that want to find out about the evaluations received by agent X, without caring about what actions he performed or what roles he plays while others are interested in obtaining a ranking of the best agents in all time playing a certain role.

In Figure 1 we have also represented the modules that handle some other trust input factors besides Reputation. Although we discuss these factors in more detail in Section 4, the modules that handle them are out of the scope of this paper.

On a first view it may seem that the system design leaves all the reputation, trust and decision making complexity on the agent side. We propose several arguments versus this perception. Firstly, this design does not change the global aspect of reputation. Reputation continues to have the same meaning, reflecting the evaluation of a group (as the system is decentralized), just that it is now more precise, better able to answer the agent's needs. We consider this approach to be more evolved than a fixed single general value to describe the overall performance of an agent when the trustor in interested just in some particular aspect. Secondly, the decision making process does not becomes more complex. The process of computing trust is not influenced by the way data is gathered. We are aware that an agent does not have too much time to make his decision, thus all the Trust Computation Module has to do is to aggregate the data every time in the same way and then compare the result to a threshold value. The modules handling trust input factors follow clear protocols, so that data acquisition does not take long time either.

The following three sections isolate each system component in order to better analyze and present its role and features.

4. TRUST

In order to decide whether to trust another or not in a given context, an agent may combine several measures, each one revealing arguments pro or against entrusting the target. This section briefly presents the main factors that can be combined in order to build trust.

Any model could use a basic value for the cases when the agent has no previous experience on an issue. This value may reflect the agent's predisposition towards trusting others. The Basic Trust Module is the one producing the value bootstrapping agent interactions.

Opinions resulting from previous interactions the agent had experienced can be analyzed and interpreted, adding an important insight to future decisions. Upon being generated, they are filed in the History Module up to a customizable storage limit, replacing the oldest ones with the most recent.

When there is no previous experience, information about role may be helpful. Generally, when not malicious, agents behave in a similar way in similar interactions when playing similar roles [4], thus, knowing what roles other agents play, may help individuals decide which are the appropriate partners for their goals. The Role Information module gathers information about the roles inside the system and discovers the ones associated to the agents to be evaluated.

As an agent's trust towards another may depend very much on how the one to be trusted is rated by the community, an important component taken into account when evaluating trust is the set of opinions the other members of the system have on the same subject. The present paper focuses on this type of information source. Section 7 describes in detail how the Aggregated Reputation module operates.

An information source that simplifies the process on the truster side is the use of trust certificates. The certificates digitally signed by a trusted authority can used by agents in order to swiftly prove their trustworthiness and competence on a situation to potential contractors. This approach may prove itself very useful for the case when there are few or no information on the target's competences on a given situation. Trust certificates are received and handled by the Certificate Analyzer Module shown in Figure 1.

In order to add information about a potential interaction partner, an agent may simply choose to ask another who has previously interacted with the target. We enumerate here some of the reasons for considering this information source costly and inefficient. When asked for his rating on a target, an agent does not restart the evaluation process (information acquisition and comparison to his standards) as if he was to decide now if to trust it or not, but provides some of the entries from his own opinion database. Those opinions may be offered as facts or they may be combined into a single satisfaction value sent to the requester as there are no norms regimenting the format of the reply. The receiver cannot tell how was the satisfaction value obtained from multiple entries, thus, it may not know how to use it. In addition, even if the requester attaches the situation description (Situation Profile) he is interested in, he has no guaranty that the information provider can interpret it (not to mention the computational overhead). This third party information approach is costly in terms of time and resources and the obtained information is little, not homogenous and it is obtained sequentially, one at a time. Finding the agents that have interacted with the target (preferable in similar situation) may be difficult in a distributed environment. Even after discovering their identities, it may be costly or impossible to communicate with them. The exchange protocol does also consume much time and the exchange data may be altered while

crossing the network. This are some of the reasons why we consider that the use of replicated repositories offer a faster and more useful information than initiating request to various individuals.

Any of the methods mentioned above may offer an insight upon the trustworthiness of another entity; still, they may themselves be trusted up to various degrees. It depends on agent's trust policy to assign different weights to each information source, and combine them into a useful result. In addition, the rigorousness when reasoning about trusting another as well as setting a threshold for cooperation depends heavily on the importance of the situation from the agent's point of view. The more important the situation, the more trust is needed to enter into a cooperative situation with another agent [15]. Different trust may be assigned to the same target, by the same evaluator if the situation differs, thus each new situation requires a re-evaluation, even for the same target agent. The agent's trust policy is implemented by the Trust Aggregation Module (in Figure 1) by using an aggregation function and establishing a threshold adequate to the importance of the situation.

5. OPINIONS

Aggregating the available information (as seen in Section 4) about a potential partner allows an agent to build a certain level of expectations, counting on which he makes the decision of trusting the other or not. The output of the interaction may, however, meet or fall under the expectancy level. It is the difference of concept between building expectations and evaluating output, when talking about trust in multiagent systems, that we want to stress in this paper.

After the interaction took place, the agent is able to emit an evaluation regarding the performance of his partner in the given situation. The description of a situation captures key elements of the context associated to the transaction. By situation we refer the tuple comprising the actual values for partner's identification, the organization he belongs to, his role inside the organization, the timestamp of the interaction, the performed action as well as some other domain-specific parameters.

We call the value that reflects the agent's satisfaction upon the output of the interaction, a Satisfaction Value. The representation of the satisfaction value highly depends on the application domain and expresses the degree of fulfillment or deception towards agent's expectation.

An opinion is a tuple consisting of the agent's identity, the tuple describing the situation and the satisfaction value assigned by agent to the output of the interaction. We name "opinion" this evaluation because it is subjective to the emitter, reflecting his own perspective relatively to a situation.

An agent's opinion expressed subsequently to the unfolding of a contract may serve multiple purposes. First, it may re-enter as feed-back in recalculating the agent trust towards future proposals. In this sense, the opinion can be associated to the target agent and serve as a record for future encounters. It can be assimilated by using a learning function to enrich the agent's general experience and ability to reason about similar situation, independent to the agent being presently evaluated. The enrichment of agent's experience may consist, for example, of readjustments made on the personal standards concerning the expected values of the variables describing the quality of an interaction.

The generated opinion may also count as input for a community module that computes a shared evaluation, reflecting the target agent's real behavior and intentions, such as the artifact described in [12]. It may also be used by aggregation functions that take multiple opinions and compute a rating value, the closest to the one the target agent "deserves". An agreement upon a situation can be extracted from the opinions of a group of agents toward the given situation, the agreement being chosen as the consensus of those opinions [11]. The model proposed in this paper uses Repositories (described in Section 6) to store the opinions from the members of a group and make them available to the group. The loop closes as set of opinions stored in a Repository may be filtered and used by any member of the group when he computes the reputation of another member.

We consider the agent system to be heterogeneous enough in what concerns preferences and evaluation standards and, in the absence of general, system-imposed norms, no one can say what is right and what is wrong when posting an opinion. Right and wrong are not previously defined and the conditions for utilizing the repository are minimal: use an account and respect the indicated format when submitting an opinion, otherwise it would not be accepted. We expect an emerging set of adjustments in each agent's behavior (the aggregation function he selects, the strictness when assigning a satisfaction value) specific to each group of agents.

In this paper, the context of each interaction is very clearly specified and employed. When talking about finding a similar situation, we do not refer to reasoning on the similarity. Instead we mean simply applying a mask on a grid (i.e. selecting the entries in the database that correspond to a pattern). If no entries correspond to the requested pattern, the agent may choose to loosen the query (by binding fewer fields to a specific value).

The only rule concerning the satisfaction value is that is has to belong to a given interval, and the agent made aware of which value represents the best score. In this paper we do not address the agent's mechanism of rating an interaction output, nor impose norm to regiment it. Each agent is free to evaluate the performance of his interaction partner according to his own rules.

6. REPOSITORIES

The Repositories defined in this model contain homogenous, multi-dimensional and non-volatile data. One of the roles of the repository is to attenuate extreme opinions. We consider it should be able to host much more entries than there could be stored on an agent's local memory. The advantage of being so vast is that it may offer a more accurate view on the way the members of a group have regarded various situations over time. In addition, the data being raw, the agent that extracts it can focus, by applying his own aggregation function, on the aspects he is interested the most – for example, he uses a higher weight for a more recent opinion.

Although the public repository could raise the costs of system maintenance it does not represent a bottleneck issue because it can be replicated. In addition security policies are simple to implement in order to control the access given the modular structure of the system.

The model we propose is conceived having in mind its application within a virtual organization. For the moment, this paper focuses

only on some key modules and functions, describing the chaining between modules. The integration of the present model into a virtual organization and its implementation using a specific technology will be covered in future work.

An important question that arises is "what entity manages the repositories?". If integrated within a virtual organization the repository management could be assigned to one of the roles within the organization. Even though it is a single owner that maintains the availability of the repositories, the model we propose offers much more flexibility and independence to the agents, as the opinions they work with and the way they process them are according to their own values and principles, and not dictated by the interests and evaluation criteria of the organization.

The entity managing the repositories should ensure a high availability and low reply time, as well as a minimum security policy that guaranties data persistence. In what concerns opinion anonymity this can be easily obtained as the public available entries in a repository do not contained any information about the agent that added his opinion. The manager of the repository can assign accounts to the agents that want to benefit from the use of this dataset. As agents are already bind to an identity in the organization (the larger system), the risk of cheating by re-entering the system with a different identity we consider to be rather low. By monitoring the activity of each account, the repository manager may detect suspicious behavior (e.g. a large number of posted opinions in a short time span). The collusion issue remains open, just like in the most part of the work found in the literature on this subject.

Once the repository system is included into the infrastructure of an organization, repository replication becomes the responsibility of the organization.

The repositories contain an important amount of data describing situation evaluations. Using a single formula to combine all this data puts the agents in no computational difficulties. Even though the entire dataset may be huge, by querying the repository in a very specific manner, the resulted list of interesting entries is not overwhelming. In addition, in each of the opinions listed, all the details describing the situation (i.e. the pattern), are the same, they were used only in the selection process, on the repository side. As the Reputation Aggregation Module (shown in Figure 1) does not process the situational details, it only deals with is a set of satisfaction values, one per opinion. On this array of numerical values, the Reputation Aggregation Module applies a more or less complex aggregation function. The aggregation function may be as simple as one that finds the arithmetical average value. It is true that for agents beneficing of higher computational capabilities, the model may be adapted and enriched so that the satisfaction value contained by an opinion is broken into multiple criteria. The system is devised having in mind an organizational environment as a main application field, making use of reasonable-enough hardware support, thus message emission-reception costs have been considered insignificant. For agent intended to be deployed in settings specific to wireless sensor networks, for example, all this costs should be re-evaluated.

The repositories themselves may offer an interface for choosing the Situation Pattern. We consider implementing them as artifacts, following the concept described in [12].

7. REPUTATION

In the model we propose, the reputation value returned by the Aggregated Reputation Module is user-oriented and shows high dynamics, following closely the requester's needs, best answering his specific questions. The resulted value represents a contextual reputation, expressing the global opinion of the community concerning the target's performance in a given situation.

While trying to evaluate the trustworthiness of a potential partner, an agent may want to find out the reputation the target agent has in the system. When on e-Bay, for example, one can easily see how others have been ranked, thus, the reputation they have, but the value alone (although relative to a scale) gives them no clue regarding target's competence. Supposing I am interested in buying a television set, how can I know if an agent that is now selling an electronic device and has a good reputation was well rated for selling good television sets or for being a good book seller, for example?

In the model we propose, an agent who tries to find reputation information is able to obtain specific, context-related opinions by querying the Opinion Repository. Being parameterized, situation can be easily filtered by the fields of interest. In order to filter situation, the model we propose uses Situation Patterns. A Situation Pattern works as a mask, by freezing one or more fields within the situation by binding them to the desired actual values, and lets the actual values of the other fields vary throughout the dataset. For example, given the structure of a situation represented by the tuple: <Agent_ID, Organization, Role, Domain, Action, Timestamp>, we can restrain the set of opinions to those concerning agentX1, the role "seller" and domain "television_sets", in no matter what organization, performing no matter what action, at no matter what time, by using the following Situation Pattern: <agentX1, _, seller, television sets, _, _, _ >.

Upon deciding what Situation Pattern the agent is interested in, the Aggregated Reputation Model sends a query to the Opinion Repository, that returns all the opinions corresponding to the requested Situation Pattern. Each opinion contains the Satisfaction Value associated with the Situation. By applying an aggregation function on the set of Satisfaction Values, the Aggregated Reputation Module is able to provide the value best reflecting the perception of the community upon the Situation of interest. The aggregation function used may be chosen from a function library, according to the agent's preferences. It may be a simple average function, a weighted mean function or any other function that combines data into a representative result. In order to reflect the dynamism of reputation and opinion aging, the weights associated to the Satisfaction Values may be determined by using an auxiliary curve function, for example SigAlfa [23]. Another example of auxiliary function that could be employed is one that counts how many opinions regard a certain Situation Pattern, thus determining if the agent will take into account or not the aggregated value. For example, an agent would not use the aggregated satisfaction values regarding a seller of a certain product unless there are more than ten opinions on this pattern. The aggregation function may also be chosen so that it returns the minimum, the maximum, a ranking or any other statistical analysis. The way an agent decides what aggregation function to use is out of the scope of this paper. The Aggregation Function Selector (shown in Figure 1) may be implemented using the artifact concept described in [12] and linked to the repository

artifact. We argue that the proposed model allows this very high flexibility thanks to the structured data used in representing opinions.

The aggregated reputation value itself will be assigned a weight and be included into the aggregation function along with other factors by the Trust Aggregation Module. The aggregation function for the trust information sources may be also be chosen from a function library, but it should be much less changing as it tightly reflects the agent's trust policy. For example, the weight for the aggregated reputation value may be influenced by the number of opinions on the requested Situation Pattern.

We call our reputation model distributed as agents can retrieve and reason on opinions issued by their peers independently to any global intermediary factor. There is no central entity to decide the criteria concerned by the provided reputation value or the aggregation function used to produce the reputation value. Opinion Repositories may be replicated through the system in order to avoid being a single point of failure.

8. FUTURE WORK

As the model presented in this paper is rather abstract and on a high-level view, there are many directions to be explored in future work. The first and most important step we will perform is to build an evaluation setup for this model. We expect the difference between the agent's expectation and the interaction output to get lower as the agent spend more in the system. This would show that the reputation system helps them better evaluate their trust in a potential interaction partner. Of course, for bootstrapping reasons, agents will set their interaction-approving threshold rather low. In time, the system will tend to become stabilized, not as a result of the convergence of opinion, but instead as a result of the way agents learn to use it in order to cover their specific informational demands.

A second direction is to integrate the model into a more specific organizational setting that may have associated a certain set of norms and see how it performs. By defining more precisely the environment we can better analyze its robustness and evaluate the possible threats.

Another interesting topic to be discussed concerns how the expectations influence the way an agent rates his collaborator. He may choose to evaluate him after the interaction against a general performance grid used for every similar situation without any influence of what he expected, or he be may choose to punish the agent whose performance did not meet the expectations. This punitive measure may be justified both because the agent lost other (maybe better) offers by choosing the agent being now evaluated and, as a social act, to accelerate the convergence of the bad rating in order to prevent others from being disappointed in the same way.

9. CONCLUSIONS

The paper proposes a system where agents are able to decide what aspect of reputation need to extract about others and are able to aggregate locally the dataset corresponding to their needs. In addition, the aggregation function is chosen by the agents themselves in order to best fit the data they have selected. As a result, agent benefit from obtaining the values reflecting the quality (or other measures) on the subset of aspects they are interested in when evaluating their peers.

As there is no system authority deciding the aggregation function and imposing a unique, universal reputation value for each agent, we may argue that the model we propose concerns dynamic and situational reputation, not only in the sense that it is recomputed each time a new opinion is issued, but for being adapted to each agent's interests and relative to a given situation.

In this paper we identify features and requirements for multi-agent systems that help increasing the agents' degree of independence, allowing them to better integrate within the system. The paper highlights both how reputation directly influences trust, and the indirect influence trust has on reputation through the use of opinions.

10. ACKNOWLEDGMENTS

This research was supported by Grant CNCSIS ID 1315, 2009-2011, and Grant POSDRU ID 7713.

This paper relies on the study done during the research mission abroad at École Nationale Supérieure des Mines de Saint-Étienne, France.

11. REFERENCES

[1] Abdul-Rahman A, Hailes S. 1997. A distributed trust model. *Proceedings of the 1997 workshop on New security paradigms - NSPW '97*, 48-60.

[2] Abdul-Rahman A, Hailes S. 2000. Supporting trust in virtual communities. In Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000. IEEE Computer Society, 9.

[3] Aberer, K., Despotovic, Z. 2001. Managing trust in a peer-2-peer information system. In CIKM '01: Proceedings of the tenth international Conference on Information and Knowledge Management, ACM Press. New York, NY, USA, 310–317

[4] Billhardt, H., Hermoso, R., Ossowski, S., & Centeno, R. 2007. Trust-based service provider selection in open environments. *Proceedings of the 2007 ACM symposium on Applied computing - SAC '07*. ACM Press. New York, NY, USA , 1375. doi: 10.1145/1244002.1244298.

[5] Bin Yu and Munindar P. Singh. 2002. An evidential model of distributed reputation management. In AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press. New York, NY, USA, 294–301.

[6] da Silva, T. V., Hermoso, R., & Centeno, R. 2009. A Hybrid Reputation Model Based on the Use of Organizations, In Coordination, Organizations, Institutions and Norms in Agent Systems IV , Springer, 111--125

[7] Dong Huynh, T., Jennings, N., Shadbolt, N. 2004. Fire: An integrated trust and reputation model for open multi-agent systems. In: ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): proceedings. Ios Pr Inc; 18.

[8] Ganesan, P., Garcia-Molina, H., Widom, J. 2003. Exploiting hierarchical domain structure to compute similarity. ACM Trans. Inf. Syst., 21(1):64–93.

[9] Giorgios Zacharia. 1999. Collaborative reputation mechanisms for online communities. Master's thesis, Massachusetts Institute of Technology, September 1999

[10] Hermoso, R., Carlos, J., & Ossowski, S. 2010. Role Evolution in Open Multi-Agent Systems as an Information Source for Trust ∗. In *AAMAS 2010*.

[11] Hermoso, R., Centeno, R. and Silva, V. 2010. Reputation-based Agreement for Agent Organisations. In Principles of Practice in Multi-Agent Systems, Springer, 624 – 631.

[12] Hubner J, Vercouter L, Boissier O. 2009. Instrumenting multi-agent organisations with artifacts to support reputation processes. *Coordination, Organizations, Institutions and Norms in Agent Systems IV*. 96 – 110.

[13] Huynh, T. 2006. Trust and reputation in open multi-agent systems. *PhD Thesis*.

[14] Koster, A., Sabater-Mir, J., Schorlemmer, M. 2009. An Interaction-oriented Model of Trust Alignment. In Proc. of the 13th Conference of the Spanish Association for Artificial Intelligence, CAEPIA, 655 – 664

[15] Marsh, S. 1992. Trust and reliance in multi-agent systems: A preliminary report. In *Proceedings of European Workshop on Modelling Autonomous Agents in a Multi-Agent World.* 94–112.

[16] Ramchurn, S.D., Huynh, T.D., Jennings, N.R.2004. Trust in multi-agent systems. Knowledge Engineering Review, 1–25

[17] Sabater J, Paolucci M, Conte R. 2006. Repage: Reputation and image among limited autonomous partners. *Journal of Artificial Societies and Social Simulation*. 9(2):3.

[18] Sabater, J., & Sierra, C. 2001. REGRET: a reputation model for gregarious societies. In Jorg P. Muller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press. Montreal, Canada , 194–195.

[19] Sarvapali D. Ramchurn, Carles Sierra, Lluıs Godo, and Nicholas R. Jennings. 2003. A computational trust model for multi-agent interactions based on confidence and reputation. In Proceedings of 6th International Workshop of Deception, Fraud and Trust in Agent Societies, 69–75.

[20] Sen S. 2002. Believing others: Pros and cons. *Artificial Intelligence*. 142(2):179-203

[21] Şensoy, M., Yolum, P. 2006. A context-aware approach for service selection using ontologies. In: Proceedings of AAMAS 2006, 931–938

[22] Urbano, J., Rocha, A.P. ,Oliveira, E. 2009. A Trust Aggregation Engine that Uses Contextual Information. In Proceedings of EUMAS2009

[23] Urbano, J., Rocha, A.P., Oliveira, E. 2009. Computing Confidence Values: Do Trust Dynamics Matter? The 14th Portuguese Conference on Artificial Intelligence, EPIA'2009, Aveiro, Portugal, October 2009

From Objects to Agents: Rebooting Agent-Oriented Programming for Software Development

Andrea Santi
DEIS, Università di Bologna
a.santi@unibo.it

ABSTRACT

The notion of *agent* more and more appears in different contexts of computer science, often with different meanings. The main acceptation is the AI (Artificial Intelligence) and Distributed AI one, where agents are essentially exploited as a technique to develop special-purpose systems exhibiting some kind of intelligent behavior. In this paper, we introduce a further perspective, shifting the focus from AI to computer programming and programming languages, seeing agents and related concepts as general-purpose abstractions useful for programming software systems in general, conceptually extending object-oriented programming with features that – we argue – are effective to tackle some main challenges of modern software development. First, we define a conceptual space framing the basic features that characterize the agent-oriented approach, then we show how such features are dealt with in practice by using a platform called JaCa. Real-world programming examples are introduced throughout to clarify the discussion.

1. INTRODUCTION

The notions of *agent* more and more appear in different contexts of computer science, often with different meanings. In the context of computer programming, agents and multi-agent systems are typically exploited – mainly in AI and Distributed AI literature – as a technique to develop special-purpose systems exhibiting some kind of individual or collective intelligent behavior [15, 20].

In this paper we discuss what we believe is a brand new and unexplored perspective, in which agents are used to define a general-purpose programming paradigm, providing a level of abstraction which can be considered an evolution of objects – as defined in OOP – and actors. So, instead of exploiting agents as abstractions to support AI techniques, here we frame the value of multi-agent programming as a general-purpose paradigm for organizing and programming software, providing features that we consider effective to tackle main challenges of modern and future software development, such as concurrency, decentralization of control, autonomy, adaptivity. For instance, concurrency has been recently raised as one of the issues that (given the growing diffusion of multi-core architectures and of the broadband Internet) will need to be necessarily tackled in mainstream programming—beside the mere academic research context where it has been studied for the last fifty years. This situation is pretty well summarized by the sentence: "The free lunch is over" as put by Sutter and Larus in [18]. Besides introducing fine-grain mechanisms or patterns to exploit parallel hardware and improve programs efficiency in existing mainstream languages, it is now increasingly important to introduce higher-level abstractions that "help build concurrent programs, just as object-oriented abstractions help build large component-based programs" [18]. We argue that agent-oriented programming – as framed in this paper – provides one such level of abstraction.

Actually, the idea of Agent-Oriented Programming is not new. The first paper about AOP is dated 1993 [16], and since then many Agent Programming Languages (APL) and languages for Multi-Agent Programming have been proposed in literature [1, 2, 3]. The objective of these works was the introduction of a novel post-OO programming paradigm to conceive complex applications: actually one can argue that in spite of this objective these works have not had a significant impact on mainstream research in programming languages and software development. We believe that this depends on the fact that (in spite of few exceptions) most of the effort and emphasis have been put on theoretical issues related to AI themes, instead of focusing on the key principles and practice of general-purpose computer programming. This is the direction that we aim at exploring in this paper.

We first define a clean conceptual space to describe the basic features of a general-purpose programming paradigm based on agent-oriented abstractions (Section 2)—based on recent results in agent-oriented meta-models and middlewares [11]. Accordingly, we provide a practical evaluation by exploiting an agent-oriented platform called JaCa (Section 3), which actually integrates two different existing agent technologies, Jason [4, 5] and CArtAgO [14]. The objective is to show how to exploit agent-oriented abstractions to conceive and develop real-world programs, and point out related outcomes and limitations. In particular, several weaknesses are related to the fact that main issues for the development of mainstream paradigms (such as OOP) have still to be suitably explored in the context of agent-oriented programming—e.g., an explicit notion of type and reusability. We discuss such issues (Section 3.5) in the context of our conceptual space, and finally in Section 4 we provide the paper conclusions.

2. AGENT-ORIENTED ABSTRACTIONS FOR COMPUTER PROGRAMMING

In this section we consider some main concepts of agent-oriented programming. While most of these concepts already appeared in literature in different contexts, our aim here is to highlight their value for framing a conceptual space and an abstraction layer useful for defining general-purpose

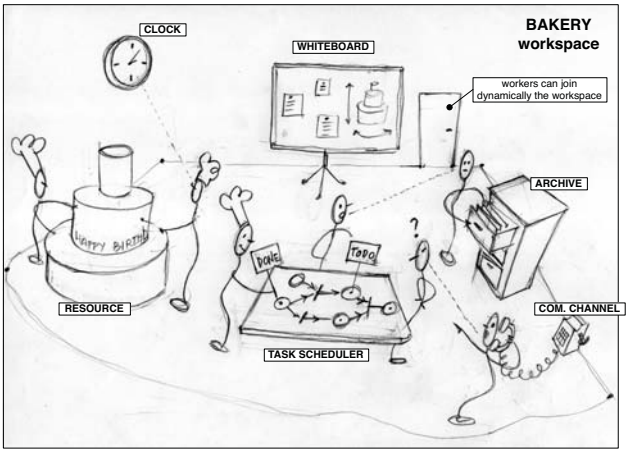


Figure 1: Abstract representation of the A&A metaphor in the context of a bakery.

programming languages. Our approach is initially conceptual, while in the next section these concepts will be exemplified in practice using concrete agent technologies.

2.1 A Background Metaphor

Metaphors play a key role in computer science, as means for constructing new concepts and terminology [19]. In the case of objects in OOP, the metaphor is about real-world objects. Like physical objects, objects in OOP can have properties and states, and like social objects, they can communicate as well as responding to communications. In the case of actors, similarly, the inspiration is clearly more anthropomorphic, and a variety of anthropomorphic metaphors influenced its development [17, 7].

The inspiration for the agent-oriented abstraction layer that we discuss in this paper is anthropomorphic too (and refers to the A&A metaphor [11]), however taking human cooperative work environments as main reference—as analyzed by Activity Theory and Distributed Cognition. Figure 1 shows an example of such metaphor, represented by a human working environment, a *bakery* in particular. It is a system where articulated concurrent and coordinated activities take place, distributed in time and space, by people working inside a common environment. Activities are explicitly targeted to some objectives. The complexity of work calls for some division of labor, so each person is responsible for the fulfillment of one or multiple tasks. Interaction is a main dimension, due to the dependencies among the activities. Cooperation occurs by means of both direct verbal communication and both through tools available in the environment (e.g. a blackboard, a clock, the task scheduler). So the environment – as the set of tools and resources used by people to work – plays a key role in performing tasks efficiently. Besides tools, the environment hosts resources that represent the co-constructed results of people work (e.g. the cake).

Following the metaphor, we see a program – or software system – as a collection of agents working and cooperating in a common environment Figure 2: on the one side, agents (like humans) are used to represent and modularize the pro-active part of the system, i.e. the part in charge to autonomously perform the tasks in which the overall labor is split; on the other side, the environment is used to represent

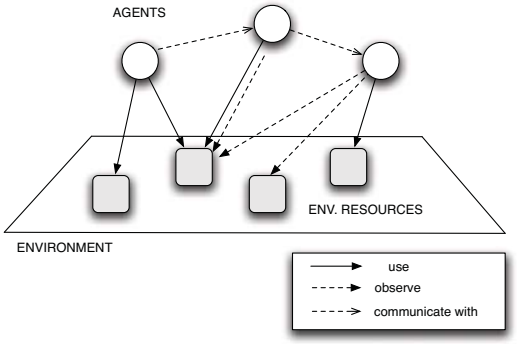


Figure 2: Abstract representation of an agent-oriented program composed by agents working within an environment.

and modularize the functionalities that can be dynamically composed, adapted and used to perform the tasks.

A main feature of this approach is that it promotes a *decentralized mindset* in programming, as also considered by Resnick in [13]. Such a mindset has two main cornerstones.

The first one is the decentralization and encapsulation of *control*: there is not a unique locus of control in the system, which is instead decentralized into agents. It is worth remarking that at this point we are still assuming a logical point of view over decentralization—not strictly related to, for instance, physical threads or processes. So, from this point of view, the agent abstraction extends the basic encapsulation of state and behavior of objects: namely, it also considers encapsulation of control over behavior, i.e., agents encapsulate the control and this can be one of the aspects concerning their *autonomy*.

The second cornerstone is the interaction dimension which includes coordination and cooperation. There are two basic orthogonal ways of interacting: direct communication among agents based on high-level asynchronous message passing, and environment-mediated interaction (see Subsection 2.4) exploiting the functionalities provided by environment resources.

2.2 Structuring Active Behaviour: Tasks and Plans

Differently from the actor approach, where decentralization and encapsulation of control are also main properties, with the agent abstraction we have an explicit high-level rationale to structure autonomous behavior by introducing the explicit notions of *task* and *plan* for agents.

The notion of task is introduced to specify a unit of work that has to be executed—the objective of agents' activities. So, an agent acts in order to perform a task, which can be possibly assigned dynamically. The same agent can be able to accomplish one or more types of task, and the *type* of the agent can be strictly related to the set of task types that it is able to perform.

Conceptually, an agent is hence a computing machine that, given the description of a task to execute, it repeatedly chooses and executes *actions* so as to accomplish that task.

If the task concept is used as a way to define *what* has to be executed, the set of actions to be chosen and performed represents *how* to executed such tasks. The first-class concept used to represent one such set is the *plan*.

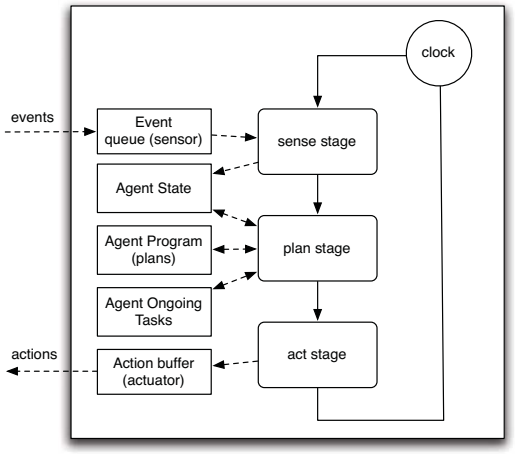


Figure 3: Conceptual representation of an agent architecture, with in evidence the stages of the execution cycle.

The agent programmer defines the behavior of an agent by writing down the plans that the agent can dynamically combine and exploit to perform tasks. For the same task, there could be multiple plans, related to different contextual conditions that can occur at runtime.

On the one side, task and plan can be used to define the contract explicitly stating what jobs the agent is able to do; on the other side, they are used (by the agent programmer) to structure and modularize the description of how the agent is able to do such jobs, organizing plans in sub-plans.

Also, this approach makes it possible to frame a smooth path in defining different levels of abstraction in specifying plans and, correspondingly, different levels of autonomy of agents. At the base level, a plan can be a detailed description of the sequence of actions to execute. In this case task execution is fully pre-defined, since the programmer is charged with the entire task specification; the level of autonomy of the agent is limited in selecting the plan among the possible ones specified by the programmer.

In a slightly more complex case, a plan could be the description of a set of possible actions to perform, and the agent uses some criteria at runtime to select which one to execute. This enhances the level of autonomy of the agent with respect to what strictly specified by the programmer.

An even stronger step towards autonomy is given by the case in which a plan is just a partial description of the possible actions to execute, and the agent dynamically infers the missing ones by exploiting information about the ongoing tasks, and about the current knowledge of its state and the state of the environment.

2.3 Integrating Active and Reactive Behaviour: The Agent Execution Cycle

Integrating both an active and reactive behavior in programming is an important programming issue – in particular for reactive systems – more generally related to the well-known problem of integrating thread-based and event-based systems [6]. Active behaviors are typically mapped on OS threads, and the asynchronous suspension/stopping/control of thread execution in reaction to an event is an issue in high-level languages. So, for instance, in order to make a thread of control aware of the occurrence of some event – to be suspended or stopped – it is typically necessary to "pollute" its block of statements with multiple tests spread around.

In the case of agents, this aspect is tackled quite effectively by the control architecture that governs their execution, which can be considered both *event-driven* and *task-driven*. The execution is defined by a control loop composed by a possibly non-terminating sequence of execution cycles. Conceptually, an execution cycle is composed by three different stages[1] (see Figure 3):

- *sense* stage – in this stage the internal state of the agent is updated with the *events* collected in the agent event queue. So this is the stage in which inputs generated by the environment during the previous execution cycle are fetched.
- *plan* stage – in this stage the next actions to execute are chosen, based on the current state of the agent, the agent plans and agent ongoing tasks; additionally, agent state is also updated to reflect such a choice.
- *act* stage – in this stage the actions selected in the *plan* stage are executed.

The agent machine continuously executes these three stages, performing one execution cycle at each logical clock tick. Conceptually, the agent control flow is never blocked—actually it can be in idle state if, for instance, the executed plan states that no action has to be executed until a specific event is fetched in the sense stage. It is worth noting this is quite similar to the control loop as found in autonomic systems and, more generally, in any reactive control system.

This architecture easily allows, e.g., for suspending a plan in execution and execute another plan to handle an event suddenly detected in the sense stage. While generally speaking this makes an agent machine less efficient than machines without such loops, this architecture allows to have a specific point to balance efficiency and reactivity thanks to the opportunity to define proper atomic actions.

2.4 "Something is Not an Agent": the Role of the Environment Abstraction

Often programming paradigms strive to provide a single abstraction to model every component of a system. This happens e.g. in the case of actor-based approaches. In Erlang [8] for instance, which is actor-based, every macro-component of a concurrent system is a process, which is the actor counterpart. This has the merit of providing uniformity and simplicity, indeed. At the same time, the perspective in which everything is an active, autonomous entity is not really always effective, at least from an abstraction point of view. For instance, it is not really natural to model as active entities either a shared bounded-buffer in producers/consumers architectures or a simple shared counter in a concurrent programs. In traditional thread-based systems such entities are designed as *monitors*, which are passive.

Switching to an agent abstraction layer, there is an apparent uniformity break due to the notion of *environment*, which is a first-class concept defining the context of agent tasks, shared among multiple agents. From a designer and

[1]Here the terms sense, plan, act are quite general, not referring to a specific AI-based technique

programmer point of view, the environment is that (non-autonomous) part of the system that provides functionalities that can be exploited at runtime by the autonomous part. In other words, the environment is both *(i)* a first-class entity for agents, to be exploited for executing their tasks, and *(ii)* a first class abstraction for programmers to define functionalities and services to be exploited. Recalling the human metaphor, the environment can be framed as the set of *resources* and *tools* that are possibly shared and *used* by agents to execute their tasks. In that perspective, a bounded-buffer, a shared data-base etc. in an agent-oriented perspective could be designed and programmed as a shared resource populating the environment where producers/consumers agents work, analogously to monitors.

2.5 Using and Observing the Environment

To be usable by agents, an environment resource provides a set of *operations* – that constitutes its *usage interface* – encapsulating some piece of functionality. Such operations are the basic actions that an agent can execute on instances of that resource type. So the set of actions that an agent can execute inside an environment depends on the set of resources that are available in that environment. Since resources can be created and disposed at runtime by agents, the agent action repertoire can change dynamically.

The execution of an operation (action) performed by an agent on a resource may complete with a success or a failure—so an explicit success/failure semantics is defined. Actions (operations) are performed by agents in the act stage of the execution cycle seen previously. Then, the completion of an action occurs asynchronously, and is perceived by the agent as a basic type of event, fetched in the sense stage. This can occur in the next execution cycle or in a future execution cycle, since the execution of an operation can be long-term. So, an important remark here is that the execution cycle of an agent *never blocks*, even in the case of executing actions that – to be completed – need the execution of further actions of other agents. This means that an agent, even if "waiting" for the completion of an action, can react to events perceived from the environment and execute a proper action, following what is specified in the plan.

Finally, aside to actions, observable events and observable properties represent the other side of agent-environment interaction, that is the way in which an agent gets input information from the environment. In particular, observable properties represent the observable state that an environment resource may expose, as part of its functionalities. The value of an observable property can be changed by the execution of operations of the same resource. A simple example is a counter, providing an inc operation (action) and an observable state given by an observable property called count, holding the current count value. By observing a resource, an agent conceptually receives the updated value of its observable properties as percepts at each execution cycle, in the sense stage. Observable events represent possible signals generated by operation execution, used for making observable an information not regarding the resource state, but regarding a dynamic condition of the resource. Taking as a metaphor a coffee machine as environment resource, the display is an observable property, the beep emitted when the coffee is ready is an observable event. Choosing what to model as a property or as an event is a matter of environment design.

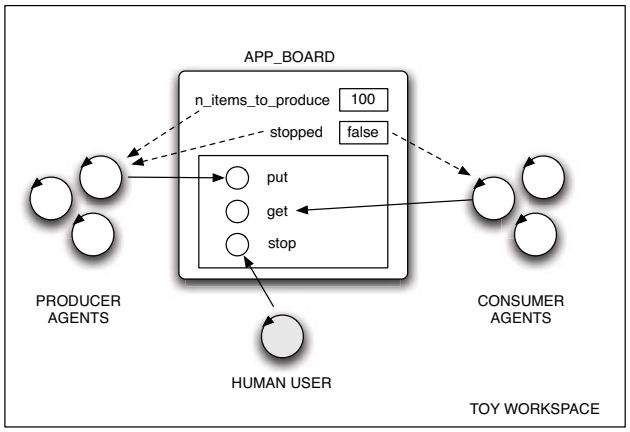


Figure 4: A toy workspace, with producer and consumer agents interacting by means of an app_board artifact.

3. EVALUATING THE IDEA WITH EXISTING AGENT TECHNOLOGIES: THE JACA PLATFORM

3.1 Overview

The aim of this section is to show more in practice some of the concepts described in the previous section. To this end, we will use existing agent technologies, in particular a platform called JaCa, which actually integrates two independent technologies: the Jason agent programming language [5] – for programming agents – and the CArtAgO framework [14], for programming the environment. Following the basic idea discussed in Section 2 - a JaCa program is conceived as a dynamic set of autonomous agents working inside a shared environment, that they use, observe, adapt according to their tasks. The environment is composed by a dynamic set of environment resources which in CArtAgO are called "artifacts"[2]. Agents – by means of proper actions – can dynamically create and dispose artifacts, beside using them.

In the following, we introduce only those basic elements of agent and environment programming which are necessary to show the features discussed at the conceptual level in the previous section. To this end, we use a toy example which is about the implementation of a producers-consumers architecture, where a set of producers agents continuously and concurrently produce data items which must be consumed by consumer agents (see Figure 4). Further requirements – which make the example more interesting for our purposes – are that *(i)* the number of items to be produced is fixed, but the time for producing each item (by the different producers) is not known a priori; *(ii)* the overall process can be interrupted by the human user in each moment. The task of producing items is divided upon multiple producer agents, acting concurrently—the same holds for consumer agents.

To interact and coordinate the work, agents share and use an environment resource, the app_board artifact, which functions both as a buffer to collect items inserted by pro-

[2]the term was inspired by Activity Theory and Distributed Cognition, where it is used to refer to any object that has been specifically designed to provide some functionality and which is used by humans to achieve their objective

```
00  n_items_produced(0).
01  !produce.
02
03  +!produce
04    <- !setup;
05       !produce_items.
06
07  +!setup
08    <- focus("app_board").
09
10  +!produce_items : not n_items_to_produce(0)
11    <- !produce_item(Item);
12       put(Item);
13       -n_items_produced(N);
14       +n_items_produced(N+1);
15       !produce_items.
16
17  +!produce_items : n_items_to_produce(0)
18    <- !finalize.
19
20  +!produce_item(Item) <- ...
21
22  +!finalize : n_items_produced(N)
23    <- println("completed - items produced: ",N).
24
25  -!produce_items
26    <- !finalize.
27
28  +!stopped(true)
29    <- .drop_all_intentions;
30       !finalize.
```

Table 1: A producer agent in Jason.

ducers and to be removed by consumers and as a tool to control the overall process by the human user. The resource provides on the one side operations (actions for the agent) to insert (`put`), remove (`get`) items and to stop the overall activities (`stop`); on the other side, observable properties `n_items_to_produce` and `stopped`, keeping track of, respectively, the number of items still to be produced (which starts from an initial value and is decremented by the resource each time a new item is inserted) and the stop flag (initially false and set to true when the `stop` operation is executed).

In the following, first we give some glances about agent programming in Jason by discussing the implementation of a producer agent (see Table 1), which must exhibit a proactive behavior – performing cooperatively the production of items, up to the specified number – but also a reactive behavior: if the user stops the process, the agents must interrupt their activities. Then we briefly consider the implementation of the app_board artifact, to show in practice some elements of the environment programming.

3.2 Programming Agents in Jason

Being inspired by the BDI architecture, the Jason language constructs that programmers can use can be separated into three main categories: *beliefs*, *goals* and *plans*. An agent program is defined by an initial set of *beliefs*, representing the agent's initial knowledge about the world, a set of *goals*, which corresponds to tasks as defined in Section 2, and a set of *plans* that the agent can dynamically compose, instantiate and execute to achieve such goals.

In JaCa agents beliefs can represent two types of knowledge:

- the agent internal state – an example is given by the `n_items_produced(N)` belief, which is used by the producer agent to keep track of the number of items produced so far.
- the observable state of the resources of the environment which the agent is observing – in the example, producer agents all observe the app_board resource, which has two observable properties: `n_items_to_produce`, representing the number of items still to be produced, and `stopped`, a flag which is set if/when the process needs to be stopped.

An agent program may explicitly define the agent's initial belief-base and the initial task or set of tasks that the agent has to perform, as soon as it is created. In Jason tasks are called *goal* and are represented by Prolog atomic formulae prefixed by an exclamation mark. Referring to the example, the producer agent has an initial task to do, which is represented by the `!produce` goal. Actually, tasks can be assigned also at runtime, by sending to an agent an achieve-goal messages.

Then, the main body of an agent program is given by the set of plans, which defines the pro-active and reactive behavior of the agent. The actions contained in a plan body can be split in two categories:

- internal actions, that are actions that are related uniquely to the internal state of the agent. Examples are actions to create sub-tasks (sub-goals) to be achieved (`!g`), to manage task execution – for instance, to suspend or abort the execution of a task – to update agent inner state – such as adding a new belief (`+b`), removing beliefs (`-b`);
- external actions, that are actions that are related to the environment, both to create/dispose/lookup/use artifacts, and to the direct communicate with other agents, to send them messages asynchronously.

Referring to the example, the producer agent has a main plan (line 03-05), which is triggered by an event `+!produce` representing a new goal `!produce` to achieve. Since the agent has an initial `!produce` goal, then this plan will be triggered as soon as the agent is booted. By means of an internal action `!g`, the main plan generates two further subgoals to be achieved sequentially: `!setup` and `!produce_items`.

The plan to handle `!setup` goal (line 07-08) exploits an external action called `focus` to start observing the app_board artifact. Then, two plans are specified for handling the goal `!produce_items`. One (line 10-15) is executed if there are still items to produce, i.e. if the agent has not the belief `n_items_to_produce(0)`. Note that the value of this belief depends on the current state of the app_board resource. This plan first produces a new item (subtask `!produce_item`), then inserts the item in the buffer by means of a `put` action[3] – whose effect is to execute the `put` operation on the resource; if this action succeeds, the plan goes on by updating the belief `n_items_produced` incrementing the number of items produced and generates a new subgoal `!produce_items` to repeat the task.

The other plan (line 17-18) is executed if there are no more items to produce: in this case the `!finalize` task is executed, which prints on the console the number of items produced by the agent.

The reactive behavior of an agent can be realized by plans triggered by a belief addition/change/removal – which cor-

[3]When executing an external action – such as `put` – it is possible to explicitly denote the artifact providing that action, in order to avoid ambiguities, by means of Jason annotations: `put(Item) [artifact_name("app_board")];`

```
00 public class AppBoard extends Artifact {
01
02   private LinkedList<Object> items;
03   private int bufSize;
04
05   void init(int bufSize, int nItemsToProduce){
06     items = new LinkedList<Object>();
07     this.bufSize = bufSize;
08     defineObsProperty("n_item_to_produce",nItemsToProduce);
09     defineObsProperty("stopped",false);
10   }
11
12   @OPERATION(guard="bufferNotFull")
13   void put(Object obj){
14     boolean stopped = getObsProperty("stopped").booleanValue();
15     if (!stopped){
16       items.add(obj);
17       ArtifactObsProperty p  =
18                     getObsProperty("n_item_to_produce");
19       updateObsProperty("n_item_to_produce", p.intValue() - 1);
20     } else {
21      failed("no_more_items_to_produce");
22     }
23   }
24
25   @GUARD boolean bufferNotFull(Object obj){
26     return items.size() < nmax;
27   }
28
24  @OPERATION(guard="itemAvailable")
25   void get(OpFeedbackParam<Object> result){
26     Object item = items.removeFirst();
27     result.set(item);
28   }
29
30  @GUARD boolean itemAvailable(OpFeedbackParam<Object> res){
31     return items.size() > 0;
32   }
33
34  @OPERATION void stop(){
35     updateObsProperty("stopped",true);
36   }
37 }
```

Table 2: The implementation of the app_board in CArtAgO.

responds, e.g., to changes in the state of the environment – and by the failure of a plan in achieving some goal. In the example, the producer agent has a plan (line 28-30) which is executed when the belief `stopped` about the observable property of the artifact is updated to `true`. This means that the human user wants to interrupt and stop the production. So the plan stops and drops all the other possible plans in execution – using an internal action `.drop_all_intention` – and the `!finalize` subtask is executed.

Finally, the producer agent has also a plan (line 25-26) to react to the failure of the `!produce_items` task, which is expressed by the event `-!produce_items`. This can happen when the agent, believing that there are still items to be produced, starts the plan to produce a new item and tries to insert it in the buffer. However, the `put` action fails because other agents produced in the meanwhile the missing items.

The semantics of the execution of plans reacting to events is defined by Jason reasoning cycle [5], which is a more articulate version of the execution cycle described in Section 2. In particular, the plan stage in this case includes multiple steps, to select – given an event – a plan to be executed. So an agent can have multiple plans in execution but only one action at a time is selected (in the plan stage) and executed (in the act stage). A detailed description of the cycle – as well as of the Jason syntax – can be found in [5].

3.3 Programming The Environment in CArtAgO

The implementation of the app_board artifact is shown in Table 2. Being CArtAgO a framework on top of the Java platform, artifact-based environments can be implemented using a Java-based API, exploiting the annotation framework. Here we don't go too deeply into the details of such API – the interested reader can refer to CArtAgO papers [14] and user manual[4] – we just introduce the main concepts that have been mentioned in Section 2.

In CArtAgO, an artifact type can be defined by extending a base `Artifact` class. Artifacts are characterized by a usage interface containing a set of operations that agents can execute to get some functionalities. In the example, the artifact app_board provides three operations: `put`, `get` and `stop`. The `put` operation inserts a new element in the buffer – decrementing the number of items to be produced – if the stopped flag has not been set, otherwise the operation (action) fails. The `get` operation removes an item from the buffer, returning it as a feedback of the action. The `stop` operation sets the `stopped` observable property to true.

Operations are implemented by methods annotated with `@OPERATION`. The `init` method is used as constructor of the artifact, getting the initial parameters and setting up the initial artifact state. For each operation, a guard can be specified, as a condition over artifact state specifying if (when) the operation is either enabled or disabled—in the example, `bufferNotFull` for `put` and `bufferNotEmpty` for `get`. Therefore, an operation can change dynamically its status, from enabled to disabled and vice-versa, according to the state of the artifact. If the guard is disabled and an agent executes that operation, the agent action is suspended.

Operations are executed in a mutual exclusive way—so only one operation can be in execution at a time inside an artifact, the other ones are suspended. In JaCa, executing an environment action for a Jason agent means that the agent plan (activity) including such action is suspended until the corresponding artifact operation has completed (i.e. the action completed). Then, the action succeeds or fails when (if) the corresponding operation has completed with success or failure. It is worth noting that, in the meanwhile, the agent execution cycle can go on, making it possible for the agent to get percepts and select and perform other actions.

Besides operations, artifacts may define also a set of observable properties (`n_items_to_produce` and `stopped` in the example), as data items that can be perceived by agents as environment state variables. Instance fields of the class – instead – are used to implement the non observable state of the artifact—for instance, the list of items `items` in the example. Observable properties are defined, typically during artifact initialization, by means of the `defineObsProperty` primitive, specifying the property name and initial value (line 08-09). Inside operations, observable properties value can be inspected and changed dynamically by means of two basic primitives: `getObsProperty` to retrieve the current value of an observable property (see, for instance, line 14 and 17) and `updateObsProperty` to update the value (line 18, 35).

Besides observable properties, an artifact can make it observable also events occurring when executing operations. This can be done by using a `signal` primitive, specifying

[4] CArtAgO is available at `http://cartago.sourceforge.net`

the type of the event and a list of actual parameters. For instance, `signal("my_event", "test",0)` generates an observable event `my_event("test",0)`. In the app_board example, to notify the stop we could generate a `stopped` signal in the `stop` operation, instead of using an observable property. Observable events are perceived by all agents observing the artifact—which could react to them as in the case of observable property change.

3.4 Using JaCa In Real-World Application Contexts

We are applying this programming model in different application domains, to stress the benefits but also the weaknesses of the approach.

One is the development of distributed applications based on Service-Oriented Architectures and Web Services in particular. In that context, agents and multi-agent systems are deserving increasing attention both from the applicative viewpoint, as an effective technique to build complex services and applications dynamically composing and orchestrating services [10], and from the foundational viewpoint, as a reference meta-model for the service-based approach, as suggested by the W3C document about Web Services Architecture[5]. To this end, programming models and platforms are needed that make it possible to build SOA/WS applications as agent-oriented systems in a systematic way, exploiting the existing agent languages and platforms to their best, while enabling their co-existence and fruitful co-operation. In that context, we devised a library of artifacts on top of the JaCa platform, enabling the development of SOA/WS applications in terms of workspaces populated by agents and artifacts. Agents encapsulate the responsibility of the execution and control of the business activities and tasks that characterize the SOA-specific scenario, while artifacts encapsulate the business resources and tools needed by agents to operate in the application domain. In particular, artifacts in this case are exploited to model and engineer those parts in the agent world that encapsulate Web Services aspects and functionalities – eventually wrapping existing non-agent-oriented code – to be used, but also changed and adapted by agents at runtime, by need. First results of this work are available here [12].

We are also investigating the approach for the engineering of advanced mobile computing applications, in particular for pervasive and context-aware computing scenarios. To this end, JaCa has been ported on the Android platform[6], enabling the development of Android applications using agent-oriented programming. The project is called JaCa-Android[7]. Actually, besides porting the technology, JaCa-Android includes a library of artifacts that allows agents running into an Android application to seamlessly access and exploit all the features provided by the smart-phone and by the Android SDK. Just to have a taste of the approach, Table 3 shows a snippet of an agent playing the role of smart user assistant, with the task of managing the notifications related to the reception of SMS messages: as soon as an SMS is received, a notification must be shown to the user. A `SMSArtifact` artifact is used to manage SMS messages, in particular this artifact generates an observable event `sms_received`

```
00  !init.
01
02  +!init
03  <-  focus("SMSArtifact");
04      focus("SMSArtifact");
05      focus("ViewerArtifact").
05
07  +sms_received(Source, Message)
08    : not (state("running") & session(Source))
09    <-  showNotification("jaca.android:drawable/notification",
10          Source, Message, "jaca.android.sms.SmsViewer", Id);
11        +session(Source, Id).
12
13  +sms_received(Source, Message)
14    : state("running") & session(Source)
15    <-  append(Source, Message).
```

Table 3: Source code of the Jason agent that manages the SMS notifications.

each time a new SMS is received. A `ViewerArtifact` artifact is used to show SMS messages on the screen and to keep track – by means of the `state` observable property – of the current status of the viewer, that is if it is currently visualized by the user on the smartphone screen or not. Finally, a `StatusBarArtifact` artifact is used instead to show messages on the Android status bar, providing a `showNotification` operation to this end. Depending on what the user is actually doing and visualizing, the agent shows the notification in different ways. The behavior of the agent, once completed the initialization phase (lines 00-05), is governed by two reactive plans. The first one (lines 7-11) is applicable when a new message arrives and the `ViewerArtifact` is not currently visualized on the smartphone's screen. In this case, the agent performs a `showNotification` action to notify the user of the arrival of a new message using the status bar (Figure 5, *(a)*). The second plan instead (lines 13-15) is applicable when the `ViewerArtifact` is currently displayed on screen and therefore the agent could notify the SMS arrival by simply appending the SMS to the received message list showed by the viewer (Figure 5, *(b)*): this is done by executing the `append` operation provided by `ViewerArtifact`.

From the example, it should be clear that for a developer able to program using the JaCa programming model, moving from one application context to another is a quite straightforward experience. Indeed, she can continue to engineer the business logic of the applications by suitably defining the Jason agent's behavior, and it only need to acquire the ability to work with the artifacts that are specific of the new application context.

3.5 Weaknesses

As said at the beginning of section Section 3, JaCa is just our first attempt to adopt agent-based abstractions and technologies for the conceiving and developing of real-world programs. Therefore the JaCa platform, as well as the programming model upon which it is based, still suffers of some limitations and weaknesses.

One of the main weakness is that the proposed approach still does not consider any explicit notion of *type*, neither for agents nor for artifacts. Therefore features like sub-classing, polymorphism, etc. are still not usable in the development of agent-based applications based on the JaCa platform. This is a quite strong limitation due to the fact that such features are the key for providing reusability of the code produced by the developers and therefore are quite essential for: (*i*)

[5] `http://www.w3.org/TR/ws-arch/`
[6] `http://developer.android.com`
[7] `http://jaca-android.sourceforge.net/`

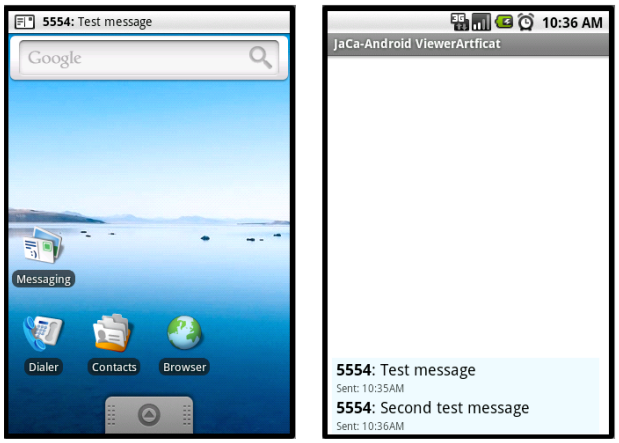


Figure 5: The two different kinds of SMS notifications: (*a*) notification performed using the standard Android status bar, and (*b*) notification performed using the `ViewerArtifact`.

the engineering of real-world applications and (*ii*) for the diffusion of the AOP as a mainstream paradigm.

Another weakness regards the lack of a seamless integration between the model/platform – in particular on the Jason side – with the Object-Oriented and Functional programming layer. Currently, for using objects/functions or for integrating any kind of software library (such as for e.g. a library for XML-manipulation), we need to use some sort of wrap mechanism for making them available when programming agents. Now we can realize this sort of wrapping in two ways: (*i*) extending the set of Jason internal actions for directly provide to the agents the required features or (*ii*) encapsulating the required object-oriented/functional-oriented code inside proper artifacts operations.

Finally, another weakness of the approach concerns the weak modularization provided by Jason plan construct. Currently the overall behavior of an agent is defined by a flat list of plans. The absence of a hierarchical structure for plans, explicitly relating plans with sub-plans, could make the understanding of complex agent behavior quite problematic.

4. CONCLUSION

Quoting Lieberman [9], *"The history of Object-Oriented Programming can be interpreted as a continuing quest to capture the notion of abstraction – to create computational artifacts that represent the essential nature of a situation, and to ignore irrelevant details"*. Following this perspective, in this paper we discussed agent-oriented programming as an evolution of Object-Oriented Programming representing the essential nature of decentralized systems where tasks are in charge of autonomous computational entities, which interact and cooperate within a shared environment. We showed in practice some of the main concepts underlying the approach by exploiting the JaCa platform, which is based on existing agent-oriented technologies—the Jason language to program agents and CArtAgO framework to program the environment. However, we believe that in order to stress and investigate the full value of the agent-oriented approach, a new generation of agent-oriented programming languages is needed, tackling main aspects that have not been considered so far in existing agent technologies – being not related to AI but to the principles of software development. This is the core of our current and future work.

5. REFERENCES

[1] *Multi-Agent Programming Languages, Platforms and Applications - Volume 1*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005.

[2] *Multi-Agent Programming Languages, Platforms and Applications - Volume 2*, Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2009.

[3] R. Bordini. A survey of programming languages and platforms for multi-agent systems. In *Informatica 30*.

[4] R. Bordini and J. Hübner. BDI agent programming in AgentSpeak using Jason. In *CLIMA VI*. Springer, 2006.

[5] R. Bordini, J. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.

[6] P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 2008.

[7] W. A. Kornfeld and C. Hewitt. The scientific community metaphor, 1981. MIT Artificial Intelligence Laboratory.

[8] J. Larson. Erlang for concurrent programming. *Commun. ACM*, 52(3):48–56, 2009.

[9] H. Lieberman. The continuing quest for abstraction. In *ECOOP 2006*, 2006.

[10] M. N. Huhns, M. P. Singh, and M. e. a. Burstein. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 2005.

[11] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2008.

[12] M. Piunti, A. Santi, and A. Ricci. Programming SOA/WS systems with BDI agents and artifact-based environments. In *MALLOW-AWESOME*, 2009.

[13] M. Resnick. *Turtles, Termites and Traffic Jams. Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[14] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*.

[15] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach (second edition)*. Prentice Hall.

[16] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[17] B. C. Smith and C. Hewitt. A plasma primier, 1975. MIT Artificial Intelligence Laboratory.

[18] H. Sutter and J. Larus. Software and the concurrency revolution. *ACM Queue: Tomorrow's Computing Today*, 3(7):54–62, Sept. 2005.

[19] M. D. Travers. *Programming with Agents: New metaphors for thinking about computation*. Massachusetts Institute of Technology, 1996.

[20] M. Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, Ltd, 2002.

Understanding ecological impacts of recreation through modeling of spatial visitor behavior

Christopher Garthe
Institute for Environmental Planning,
Leibniz University Hannover,
Herrenhäuser Str. 2
30419 Hannover, Germany.
+49 (0)30 77008019
christopher.garthe@gmx.de

ABSTRACT
This paper presentes the concept and development of a simplistic model for the spatial visitor behavior around campsites. After describing the model, first results of a local sensitivity analysis comprise general hypothesis on the relationship between environmental characteristics and the visitor behavior.

Categories and Subject Descriptors
I.6.5 [**Simulation and Modeling**]: Model Development, I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms
Human Factors, Management

Keywords
Simulation of recreational use, agent based modeling, protected area management, spatial visitor behavior, visitor flows

1. INTRODUCTION

1.1 Problem statement
Nature and National parks are not only areas protected for nature conservation but at the same time attractive regions for recreationists and destinations for tourists. The increasing importance of recreation and tourism in general and recreational activities in nature in particular (Opaschowski 1999) has led to a rising recreational pressure on protected areas (Eagles et al. 2002). Thus, protected area managers are regularly confronted with this conflict of objectives.

To come to an informed decision about recreational use in protected areas knowledge about the effects of this use on the protected habitats are crucial. These ecological effects of recreation are studied in the field of recreation ecology since the 1960s. The impacts could be differentiated between the altered ecological components or the type of use and the type of impact respectively. Impacts alter amongst others soil, vegetation, fauna and water. Types of impact comprise the development of informal trails, trampling, contamination through feces, disturbance of wildlife etc. (Cole 1986; Leung, Marion 2000).

The ecological components as well as the protected habitats differ greatly in extent and with regard to their sensitivity and resistance to these impacts as well as their ability to recover. Besides the status of protection of the habitats, these characteristics have to be considered by the park management when deciding on the level of recreational use of an area.

It becomes obvious that it is crucial to know where the recreational use takes place within a protected area, in order to come to an informed management decision which does justice to nature conservation as well as visitor management.

1.2 Existing research
Currently there are three major research approaches to cope with the problem described. The most direct approach is to incorporate a spatial perspective in recreation ecology studies; e.g. analyzing the observed effects for spatial patterns (e.g. Leung 1998; Cole, Monz 2004; Kangas 2007).

A second approach is to monitor the visitors in the area. Apart from observation or self-registration approaches various techniques have been developed comprising sensor mats, infrared visitor counts, video cameras and GPS-tracking of visitors (Arnberger, Hinterberger 2003).

A third approach is to model the movement and distribution of visitors within an area. The field of recreation use simulation has started in the 70s. Cole (2005) and Gimblett (2008) give a good overview of the development of the field as well as recent approaches. Complex models that are currently used to simulate visitor behavior in protected areas comprise two verified models from Itami et al. (2003) and Jochem et al. (2006).

1.3 Research deficit
The two models from Itami and Jochem are adjusted to a large scale comprising a complete protected area. They are not explicitly designed to simulate visitor flows on a smaller scale like specific recreation sites. Other work on modelling of microscopic pedestrian behavior has been done (e.g. Hoogendoorn 2002), but mostly under other circumstances, for which reason these results do not seem to be applicable on the situation of spatial visitor behavior in nature reserves. However, habitats and ecological components often change on a smaller spatial scale; protected areas, especially in middle Europe, consist

Cite as: Christopher Garthe (2010): Understanding ecological impacts from recreation through modeling of spatial visitor behavior. In: *Proceedings of the EASSS 2010.*

of a mosaic often at a small scale. Thus, to further understand the ecological impacts of recreation the model presented in this paper aims at simulating spatial visitor behavior on a smaller scale.

Furthermore, most research approaches in this field look at visitors moving on trails, therefore using linear-based models. Behavior rules for agents are based on route choice decisions. When studying small-scale areas, e.g. the vicinity of picnic sites or campsites, visitors usually do not move on trails. Therefore, a model has to employ a grid-based approach that also acknowledges decision processes, different from models looking on spatial behavior, which is confined to infrastructure like streets or trails.

Up to this day, the author knows of no model that looks at spatial visitor behavior in natural surroundings on a small scale with visitors moving freely in the vicinity of an initial point.

1.4 Objective

The objective of this research is to gain understanding on the spatial movement of visitors in the vicinity of campsites through the development of an agent-based model. The model is not designed to predict concrete visitor movement at specific sites, but aims to identify the crucial parameters and driving forces, which influence this visitor movement.

The results of the model could inform protected area managers to better plan and implement location-specific visitor management actions. Thus, the model contributes to an effective protected area management that acknowledges both, nature conservation needs and demand for recreation.

Furthermore, the model results could be an important input for future model developemnts that aim to simulate location-specific visitor movement at recreational sites.

2. METHODS

To simulate the processes of recreational use in the vicinity of campsites, an agent-based model was developed. An overview of the model will be given by using the standard protocol for describing agent-based and individual based models (Grimm et al. 2006). Input into the model development process, e.g. relevant parameters and processes, was gathered from literature on recreational visitor behavior as well as recreation use simulation (e.g. Cole 2005; Gimblett 2008).

After the development process, the model was programmed using the software environment Netlogo (Sklar 2007).

To gain preliminary results about the importance of parameters, a local sensitivity analysis was executed (Railsback 2010). Based on the reference parameter set, all parameters were varied by +5% and -5%. For all parameter settings, the model was run 50 times to estimate the mean of the output parameters. The sensitivity was calculated using the following equation:

$$S^{+} = (C^{+} - C) / (dP / P)$$

S^{+}: Sensitivity (for value P+dP)

C^{+}: mean output (for value P+dP)

C: mean output (for value P)

dP: amount by which P is varied (here: 5%)

P: reference value of the parameter

3. MODEL DEVELOPMENT

This section should clarify the purpose of the model as well as the design concepts used during development. It also describes in detail the entities and the programmed processes of the model.

3.1 Overview

3.1.1 Purpose

The model was designed to understand how specific characteristics of a campsite and its surroundings influence the movement of visitors in the vicinity of this site. The model tries to simulate the movement of the visitors in a rather simplistic way and thus revealing the importance of various parameters. Crucial questions guiding the development process were: How do environmental characteristics influence the distance visitors move away from their campsite?

3.1.2 Entities, state variables and scales

Entities of the model are individuals, in this case visitors, and square patches of the model world. The model comprises three types of patches: the campsite, attractions around this site and the circumjacent environment. The campsite is the starting point of the visitors. The attractions are points of interest for the visitors in the vicinity of the site, e.g. a source for drinking water, a spot to observe wildlife or a vista over a valley. The attractions are described by its attractiveness. The visitors are described by their location, which is the patch they are on. They also have three state variables, which are important for their behavior: i) time budget, which is the time they have before the start to move back to their campsite; ii) time for attractions, a amount of time they will spend on an attraction visited, iii) interest in attractions, which will determine when they will start to move towards an attraction nearby.

The model world is represented by a square with an edge length of 200 patches. Model runs last until all visitors have returned to the campsite. This, of course, is determined by the starting value of their time budget. As the model is generic, the size of each patch, the overall extent of the model world and the length a time step represents are not specified.

3.1.3 Process overview and scheduling

After the individuals are born, they start to roam in the vicinity of the site. Each individual moves once on each time step. If they are on an attraction, they stay there until their time for attraction is up; if it is up, they start to roam again. As the individuals do not interact with each other, it is not important in which order they move.

Each time step the time budet of all individuals is reduced by one. If the time budget of an individual is zero, its starts to move back to the campsite.

3.2 Design concepts

The following design principles should ensure to produce model results that describe the extent of spatial visitor movement around campsites.

The spatial movement of the individuals and thus, the values of the output parameters ***emerge*** from the adaptive behavior of the individuals as well as the characteristics of the campsite, the attractions and the environment.

The ***adaptive behavior*** of individuals concerns their reaction to the model world, more precisely to the values of variables of the different patches. This adaptive behavior is based on a few simple empirical rules: i) visitors want to move to 'attractions' in the vicinity of the site, ii) which attraction they want to visit is determined by the attractiveness as well as the distance of the specific attraction, iii) once they found an attraction, they want to spend some time there, iv) visitors want to move back to the campsite after a certain amount of time.

Table 1: Results of the local sensitivity analysis

Process and parameter	Parameter description	Reference value	Sensitivity S^+	
			Maximum distance from campsite	Mean distance from campsite
Environment				
$Attr_{Num}$	Number of attractions in the vicinity	10	15.65	9.45
$Attr_{Attractiveness}$	Mean Attractiveness of the attractions	50	17.72	14.73
Visitors				
Vis_{Num}	Number of visitors	10	14.78	-2.39
$Vis_{Time\text{-}Budg}$	Time visitors have before they return the campsite	100	-4.71	4.54
$Vis_{Time\text{-}Attr}$	Time visitors spend at the attractions	5	-8.08	-7.13
$Vis_{Interest}$	Threshold defining when visitors are moving towards the attractions	20	27.24	1.71

Individuals also change their adaptive traits over time, as the model comprises a very simple ***learning*** mechanism: Individuals have a history of visited attractions, so they do not move to the same attraction twice.

Although ***sensing*** of the individuals is the basis for their adaptive behavior, it is programmed through an indirect equation, which uses the attractiveness of the attractions, the distance of individuals to the attractions and the interest for the attractions. Thus, individuals are assumed to know all these variables as well as their own time budget.

Although much work has been done on the effects of ***interactions*** on the individual behavior of agents (e.g. Hoogendoorn 2007), interaction does not seem to be a major factor in this model development.

As this is model does not simulate the visitor movement in a specific area, but is a generic model, ***stochasticity*** is used to setup the model world. In addition, stochasticity is used to represent the direction of movement individuals, after they visited an attraction.

For ***observation*** of the model outome two output parameters are defined: the maximum distance of any individual from the campsite and the mean distance from the campsite of all individuals.

3.3 Details

During ***intialization*** of the model, the values of varaibles are asigned to the patches. The number of attractions is chosen. Stochasticity influences the disposition of the attractions and the attractiveness of attractions. The mean of the attractiveness of the attractions is defined. The number of visitors could be defined and their initial location is the campsite, a specific patch in the middle of the model world. The variables time budget, time for attractions and interest for attractions are defined during initialization.

As this generic model does not simulate a specific recreational site, and as the environmental values are assumed to be constant there is no ***input data***.

There are only two ***submodels:*** roaming and visiting attractions. When roaming, individuals start to move in a random direction. The interest for attractions serves as a threshold when the individuals start to consider moving towards an attraction: if the value of interest is bigger than the attractiveness of an attraction divided by the distance to it, the individuals 'sense' it as a relevant attraction. The individual startes to move towards the attraction, which value of attractiveness divided by distance is the highest.

This simple behavioral rule should reflect the complex decision processes in this situation, aknowledging the two most important factors for the decision: the importance, or attractiveness, of the site for the visitor as well as the distance to it.

When an individual has moved on an attraction patch, it starts the submodel visiting. The individual stays on this patch and each time step the time for attractions of this individual is reduced by one. When the time for attractions is equal or below zero, the individual starts to roam again, resets his time for attractions on the original value and adds this attraction to the list of visited attractions.

Individuals return to the campsite after the time budget is equal or lower than zero. When all individuals have reached the campsite the model run is finished.

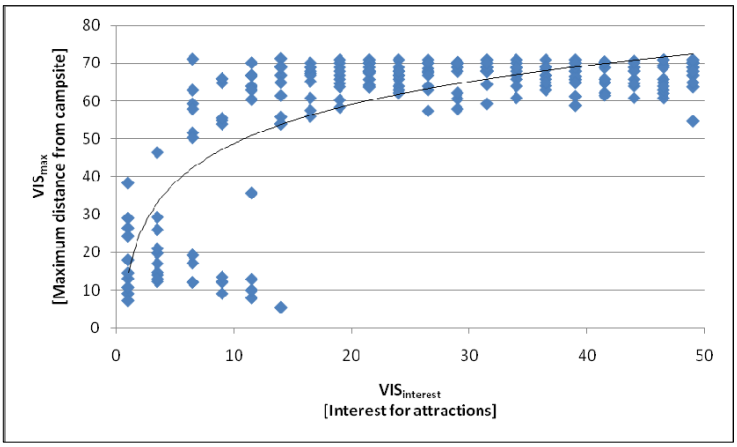
4. PRELIMINARY RESULTS AND DISCUSSION

First model runs were used for testing and parametrizing the model and submodels. A local sensitivity analysis was executed for the two output currencies maximum distance from campsite and mean distance from campsite. Table 1 shows the parameters and the associated sensitivity for the output currencies.

Looking at table 1, it becomes obvious that for the output currency 'maximum distance from the campsite' the individuals' interest for attractions is especially important. Thus, the relationship between interest and maximum diatance was further analyzed. Figure 1 shows the results of ten model runs for each step of this parameter. The relationship is best described by a logarithmic function.

As the decision process of the individuals whether to move towards an attraction or not is defined by the interest parameter, these results suggest that more input is needed to increase the quality of knowledge to calibrate this parameter.

Figure 1: Relationship between maximum distance of visitors from campsite and their interest for attractions



5. LIMITATIONS

The model presented is a generic model that does not aim to simulate visitor behavior in real world environments. Because of the simplistic assumptions for the initialization of the model world as well as the behavioral rules for visitors, the model has various limitations.

A major limitation is the input data used to develop the model. As the data was only generated by literature research, detailed empirical data is yet to be implemented in the model. A next step will be to conduct interviews of protected area managers, unobstrusive observations of visitor behavior as well as surveys of objectives and characteristics of visitors (Millonig 2010). The obtained data could be used to further calibrate the model. This seems to be sepecially important to better programm the decision process of the individuals through the interest parameter.

The model presented does only include parameters that drive visitors to leave the campsite and visit attractions. Other factors that drive visitors to stay at the campsite might also be worthwhile to incorporate into the model. So crucial questions for further model devleopment could be: Which factors detain individuals from visiting attractions in the vicinity? How does the infrastructure of the campsite influence the spatial visitor behavior?

Another limitation is that interactions between visitors are not accounted for. However, with an increasing number of visitors, interactions could become important as visitors might want to move to attractions with fewer other visitors. So with strongly increasing group sizes staying at the campsite, interactions could become an important factor for the spatial use of the area.

A further limitation is the small extent of the model world, which was used due to limited computer resources to run the model. To simulate visitor movement in a greater area around the campsite the extent of the model world has to be increased.

6. OUTLOOK

The model presented could be used to further investigate the general relationship between the use of campsite surroundings and the parameters included in the model. By further analysis of the model, hypothesis could be geenrated that could be tested in future experiments or recreation ecology research.

To make more applied use of the model outcomes, it would be desirable to expand the model in order to simulate visitor behavior or movement at a specific recreational site.

To achieve this, more parameters have to be incorporated into the model, it has to become more complex with regard to the behavioral processes of the individuals and the input of detailed environmental data is essential.

7. REFERENCES

[1] Arnberger, Arne; Hinterberger, Beate (2003): Visitor monitoring methods for managing public use pressures in the Danube Floodplains National Park, Austria. In: Journal for Nature Conservation, Is. 11, N. 4, pp. 260–267.

[2] Cole, David N. (1986): Resource impacts caused by recreation. In: The President's Commission on Americans Outdoors (Ed.): A literature review. Washington: The President's Commision On American Outdoors (INT 4901 Publication #166), INT 4901 - Publication#165, pp. 1–11.

[3] Cole, David N. (Ed.) (2005): Computer Simulation Modeling of Recreation Use. Current Status, Case Studies, and Future Directions: U.S. Department of Agriculture, Forest Service Publications (General Technical Report, RMRS-GTR-143).

[4] Cole, David N.; Monz, Christopher A. (2004): Spatial patterns of recreation impact on experimental campsites. In: Journal of Environmental Management, Is. 70, N. 1, pp. 73–84.

[5] Eagles, Paul F. J.; Haynes, Christopher D.; McCool, Stephen F. (2002): Sustainable tourism in protected areas. Guidelines for planning and management. Gland: IUCN The World Conservation Union (Best Practice Protected Area Guidelines Series, 8).

[6] Gimblett, H. Randal; Skov-Peterson, Hans (Eds.) (2008): Monitoring, simulation, and management of visitor landscapes. Tucson: University of Arizona Press.

[7] Grimm, Volker; Berger, Uta; Bastiansen, Finn; Eliassen, Sigrunn; Ginot, Vincent; Giske, Jarl et al. (2006): A standard protocol for describing individual-based and agent-based models. In: Ecological Modelling, Is. 198, N. 1-2, pp. 115–126. Online: http://www.sciencedirect.com/science/article/B6VBS-4K606T7-3/2/1dad6192bec683f32fce6dee9d665b51.

[8] Hoogendoorn, S.; Bovy, P.; Daamen, W. (2002): Microscopic pedestrian wayfinding and dynamics modelling. In: Schreckenberg, M.; Sharma, S. (Eds.),

Pedestrian and Evacuation Dynamics, pp. 123–155. Springer.

[9] Hoogendoorn, S.; Daamen, W. (2007): Microscopic Calibration and Validation of Pedestrian Models: Cross-Comparison of Models Using Experimental Data. In: Schadschneider, A.; Pöschel, T.; Kühne, R.; Schreckenberg, M.; Wolf, D. E. (Eds.): Traffic and Granular Flow'05, pp. 329–340. Springer Berlin Heidelberg.

[10] Itami, Robert; Raulings, Rob; MacLaren, Glen; Hirst, Kathleen; Gimblett, Randy; Zanon, Dino; Chladek, Peter (2003): RBSim 2: simulating the complex interactions between human movement and the outdoor recreation environment. In: Journal for Nature Conservation, Is. 11, N. 4, pp. 278–286.

[11] Jochem, Rene; Pouwels, Rogier; Visschedijk, Peter A. M. (2006): MASOOR: The Power to Know. A Story About the Development of an Intelligent and Flexible Monitoring Instrument. In: Siegrist, Dominik; Clivaz, Christophe; Hunziker, M.; Iten, S. (Eds.): Exploring the Nature of Management. Proceedings of the Third International Conference on Monitoring and Management of Visitor Flows in Recreational and Protected Areas. Switzerland, 13-17 September 2006. Rapperswil, pp. 347–350.

[12] Kangas, K. Sulkava P. Koivuniemi P. Tolvanen A. Siikamäki P. Norokorpi Y. (2007): What determines the area of impact around campsites. A case study in a Finnish national park. In: Forest Snow and Landscape Research, Is. 81, N. 1-2, pp. 139–150.

[13] Leung, Yu-Fai (1998): Assessing and evaluating recreation resource impacts. spatial analytical approaches. Unpublished thesis. Blacksburg. Virginia State University, Forestry.

[14] Leung, Yu-Fai; Marion, J. F. (2000): Recreation impacts and management in wilderness. A state-of-knowledge review. In: Cole, David N.; McCool, Stephen F.; Borrie, William T.; O'Loughlin, J. (Eds.): Wilderness science in a time of change - Vol. 5. Wilderness ecosystems, threats, and management. Wilderness Science in a Time of Change Conference, MISSOULA, MT. May 23-27, 1999. Ogden, UT: U.S. Department of Agriculture, Forest Service Publications (Proceedings, RMRS-P-15-VOL-5), RMRS-P-15-VOL-5, pp. 23–48.

[15] Millonig, A.; Gartner, G. (2010): A Multi-Method Approach to the Interpretation of Pedestrian Spatio-Temporal Behaviour. In: Klingsch, W. W. F.; Rogsch, C.; Schadschneider, A.; Schreckenberg, M. (Eds.): Pedestrian and Evacuation Dynamics 2008, pp. 563–568. Springer Berlin Heidelberg.

[16] Opaschowski, H. W. (1999): Umwelt, Freizeit, Mobilität: Konflikt und Konzepte. Opladen (Freizeit- und Tourismusstudien, 4).

[17] Railsback, Steven F.. & Grimm, Volker (2010): A Course in Individual-based and Agent-based Modeling. Princeton University Press. Forthcoming.

[18] Sklar, Elizabeth (2007): Software review: NetLogo, a multi-agent simulation environment. In: ARTIFICIAL LIFE, Is. 13, N. 3, pp. 303–311.

A Study of Resource Discovery in Open Multi Agent System and Grid Environment

Muntasir J. Al-Asfoor

School of Computer Science and Electronic Engineering

Essex University, Wivenhoe Park, Colchester, UK CO4 3SQ

mjalas@essex.ac.uk

ABSTRACT

In this paper, we have discussed the resource discovery in open distributed systems (Grid). A study of the related work and the research's directions as well as the advantages and disadvantages of each method have included. Based on intensive literature reviews, we proposed architecture for an open distributed system which facilitates the characteristics of Grid networking and the use of multi agent techniques for the purpose of resource sharing. The main objective of the proposed method is to improve the resource discovery performance by decreasing the request answer time as well as keeping the quality of matching.

1- INTRODUCTION

In the early-to-mid 1990s, there were numerous research projects underway in the academic and industry communities that were focused on distributed computing. One key area of research focus was on developing tools that would allow distributed high performance computing systems to act like one large computer [1]. Analogous to an electricity power grid, Grid computing views computing, storage, data sets, expensive scientific instruments and so on as utilities to be delivered over the Internet seamlessly, transparently and dynamically as and when needed, by the virtualization of these resources [19]. A Grid is a very large scale, generalised distributed system that can scale to Internet-size environments with machines distributed across multiple organisations and administrative domains [3].

The fundamental problem in Grid computing is the discovery of resources in such a heterogeneous computing environment as well as the dynamicity of resource enrolment and leaving processes. Resource discovery is a key concept in the distributed Grid environment; it defines the process of locating resource's providers and retrieving resource's descriptions [23].

Grid computing enables virtual organisations to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control and an existing trust relationship [1].

A **virtual organisation** represents an institution that offers one or more Grid services. Virtual organisations can span from small corporate departments that are in the same physical location to large groups of people from different organisations that are spread out across the globe. It can be large or small, static or dynamic [1, 12].

A resource is an entity that is to be shared. It can be computational such as a personal digital assistant, laptop, desktop, workstation, server, cluster, and super computer or a storage resource such as a hard drive in a desktop, (redundant Array of Inexpensive Disks), and a terabyte storage device. Bandwidth is yet another resource that is used in the activities of the virtual organisations.

An absence of a central location and central control imply that the Grid resources do not require a particular central location for their management [1].

Nowadays, the syntax based and name lookup matching techniques used by web search engines lack the ability to discover a service or resource according to the meaning of the term that represents them. They employ a simple string matching to compare two terms with only two possibilities: either finding the exact match or not. Accordingly, these techniques are not suitable for Grid environments where different users might describe the same term in a different ways.

In the Grid environment where so many different implementations are available, the need for semantic matching based on a defined ontology becomes increasingly important. In service discovery, an ontology description is very useful in supporting a customised service discovery process [18]. In order to support the discovery of relevant information with respect to a target request, resources need to be described in a way that is understandable and usable by the networked organisations. An ontology defines a common vocabulary for those who need to share information in a domain. In other words, an ontology constitutes common ground for those wishing to engage in meaningful interaction [9].

The Grid and agent communities have both developed concepts and mechanisms for open distributed systems [10]. Agents with different expertise and computational resources which may be distributed over a network can come together and coordinate to solve a complex task [9].

The rest of the paper is organised as follows: section (2) presents an intensive literature review of the related works in the fields of resource sharing in Grid environment, semantics and multi agent systems. The problem description and research objectives have been explained in section (3). Section (4) discusses conceptual system architecture as well as a high level description of the system components. Section (5) suggests two implementation scenarios based on two different ways of dealing with the problem. Proposed contributions have been discussed in section (6) and conclusions and future works have explained in section (7).

2- LITERATURE REVIEW

Researchers in the industry and academia have developed many frameworks and tools for managing resource description and discovery in Grid environments. The Globus toolkit [21] Monitoring and Discovery System (MDS4) has developed a set of monitoring and discovery techniques for services and resources in a distributed environment. It makes use of WSRF (Web Services Resource Framework) but in a centralised way. With this centralised way for managing the monitoring and discovery of resources, any problem that might occur in the node where all the indexing information is stored, will lead to the halt of the entire system.

Grid resource brokering algorithms are presented by [7] for enabling advance reservations and resource selection based on performance predictions. They suggested a decentralized broker to select computational resources based on actual job requirements, job characteristics, and information provided by the resources, with the aim to minimize the total time to delivery for the individual application. The main drawback of this method is how to determine the job requirement which needs an intervention from the user to fix the job requirements.

Erdil et. al [8] employed gossiping protocols for resource discovery. They made use of three types of gossiping protocols, in which each node can disseminate its resource information to the neighbouring nodes, and then they disseminate the information to their neighbours and so on. In this scenario, each node could be a requester or a provider. Accordingly, a resource is characterised by a resource descriptor triple (Type, Units, and time Slots); A request is characterised by the same three parameters. No semantic matching is employed in this method and a simple type matching is used to match the request and advertisement.

Another philosophy to deal with Grid resource management is used by [15]. They proposed a Grid service platform that dynamically provisions resources for both interactive and batch applications to meet their Quality of Services (QoS) constraints while ensuring good resource utilisation. In this research, the authors believed that relying on an agent-based approach for resource management will allow a more flexible, robust and scalable solution. The research is based on a multi-agent approach to capture the heterogeneity of hosted applications (in terms of allocation semantics) as well as to provide good dependability. In this research, the authors focused more on computational aspects like fault tolerance rather than matching and description.

A Virtual Organisation (VO) infrastructure is presented by [11]; according to this infrastructure, the resource broker needs to deal with an organisation instead of an individual user. Each VO then enables its users to use the resources according to VO-specific policies. The research found that it is hard to understand the tasks behaviour from the application level. Their approach was to use bounds within which allocations could be supported. However, this approach can introduce inefficiencies, which may decrease utilization for the resource provider.

A Grid environment was studied in [4]. They have utilised semantic matching in their work with the aim of making the Grid system more transparent in the application level. In this research the authors tried to achieve some sort of scalability by imposing homogeneity of resources on a virtual organisation. Using this technique is on the opposite of the idea of Grid resource sharing with main principle for resources to be heterogeneous.

Research work of building a Grid service discovery middleware with semantic web technologies is proposed by [13]. In this work, they employed the semantic web discovery technologies in the field of Grid resource discovery by adding machine-processable explicit knowledge into the interaction between pervasive devices and Grid services. The system uses a profile model to describe Grid resources. According to their results, the system has showed an increase in service discovery time compared to the traditional service discovery mechanism while a significant improvement in the system function has been obtained.

Amaranth et. al [2] proposed a semantic component in conventional Grid architecture to support ontology-based representation of Grid metadata and facilitate context-based information retrieval that complements Grid schedulers for effective resource management. They have proposed a domain-based ontology template to describe all the possible resources which means all the requesters and providers have to use this template. In a highly heterogeneous environment like the Grid, users or agents might use different ontologies to describe the resources and might not use the same ontology template. Furthermore, the system uses a centralised semantic repository which makes the system fragile to any error may occur in the node that hold the repository.

Somasundaram et. al [22] Addressed the need of semantic component in the Grid environment to discover and describe the Grid resources semantically by introducing a knowledge layer above the resource broker. In this research the system relays on Gridbus[1] to describe and discover the resources. The authors assumed that the description of resources will base on a predefined ontology template to capture the resource heterogeneity while practically it is not preferable to force the providers to use a predefined ontology template. Each provider may use different ontology template or description to describe the same terms.

Resource information integration and searching over the semantic small world[2] has been proposed in [17]. In this paper, the system uses lightweight characteristics of each node by using an Ontology Signature Set (OSS) so the similarity is measured by matching these small set of OSS. This approach forms some sort of distributed ontology matching approach but at the same time it will cause a high heterogeneity among nodes in the network by employing different ontology for each node.

A semantic supported and agent-based decentralised Grid resource discovery mechanism is proposed by [14]. The algorithm allows individual resource agents to semantically interact with neighbouring agents based on local knowledge and to dynamically form a service chain to complete a predefined task. In this research, the system is modelled

[1] See http://www.cloudbus.org/broker/

[2] The small-world networks exhibit special properties, namely, a small average diameter and a high degree of clustering, which make them effective and efficient in terms of spreading and finding information [16].

as a network graph consisting of n nodes or agents. The authors assumed that the node should have the ability to find neighbouring nodes to form a resource chain.

Different techniques were used by [25] for semantic discovery in a Grid environment. They considered the system with super nodes that hold resources. Users can locate a resource by performing a desired web service query. The system can help the user to search the web services which match his requirement and then notify that user. The paper uses Profile matchmaking techniques to decide the degree of matching two concepts.

Castano et. al [5] have proposed an algorithm for resource discovery based on the idea of considering both linguistics features of the concepts in the ontology as well the semantic relations among concepts in a peer ontology. They made use of the H-MATCH algorithm to compute the degree of similarity between two terms represent two concepts. The first step in this algorithm is to use the WordNet thesaurus paths to compute the Linguistic Affinity (LA). Secondly, they compute the Relational Affinity (RL) for the concepts relations and properties based on weights taken from the ARTEMIS [24] framework. Finally, they used both the LA and RL to compute the Contextual Affinity (AC) which represents the degree of similarity between the two terms. In this system the accuracy of the results is based on WordNet thesaurus which might not be efficient to find the degree of similarity between two concepts from different domains or using different vocabularies. Furthermore, according to their evaluation the system has showed some increasing in the query answering time when compare with the conventional matching methods.

As shown in the literature review many research works have dealt with the topic of resource management in Grid environment. These researches have focused partially on the problem with each one see the problem from different points of view.

Some researchers have seen the problem from the network performance point of view focusing on different layers of networking to improve the overall network functionality. In this case, the efforts were focusing on network aspects like (network topology, network technology) with the aim of improving factors like (bandwidth, delay, etc.). This research direction is not related to our objectives.

Other researchers have focused on the jobs requirements prediction in order to develop mechanisms for advance resource reservations. Using these techniques required a user intervention to provide information about the jobs like requirements and characteristics.

Our concern is regarding the semantic resource discovery in open distributed systems (computational Grid) environment with the aim of improving this process and keeping the characteristics of the system, more specific the heterogeneity and dynamicity. Many researchers have worked on this topic but partially. Some directions are in contrary with the philosophy of open distributed systems like homogeneity and centralisation. Furthermore, some systems are using non-semantic matching techniques like type matching and profile matching.

3. The Problem Statement and research objectives

In open distributed systems where the resources are geographically distributed across heterogeneous nodes, an efficient resource discovery method has become an essential requirement. Nowadays, many resource discovery systems are using the simple keyword (syntactic) matching methods to match the user's request with the provided resources. Using this matching method the result would be either exact matching (i.e. either the request matches the advertisement exactly) or not.

As the resources are highly heterogeneous and the way the providers advertise their resources might vary from the one the requester uses; many researchers in the academia and industry have built frameworks to discover the resources semantically depending on the functionalities of the resources rather than simple name matching methods. Accordingly, these methods have dealt partially with the problem as shown in section (2). As the open distributed systems are heterogeneous and dynamic, the main factors which play a crucial role are:

- The quality of matching (i.e. how relevant the advertisement is to the request), for this purpose many researchers have employed the ontology description along with semantic reasoners. In this case, the main issues are the matching of a request and advertisements which are described ontologically using different description languages.
- The response time: in a highly dynamic environment like open distributed systems time is the most important factor. It is obvious that semantic discovery is more time consuming since it involves more checking and computation than simple keyword matching.

According to these two factors, the question now is: how to achieve as high quality of matching as possible while keeping the required time as short as possible?

To deal with the question has mentioned above the research will address the trade-off between the quality of the matching algorithm and the required time for completion. Furthermore, the research will compare between the linguistics and contextual matching methods according to the quality and time factors. To achieve this goal, the research will follow two directions: the first direction is to improve the request and the advertisement description to make it more compatible with search and matching algorithms.

Afterwards, the research will address the case of "no match" (i.e. if the matching process returns false "no match"). By employing multi-agent system techniques, the research will exploit the agent's capabilities to reform the requests in a way that decreases the user's requirements via negotiation over less important requirements.

4. The Proposed Architecture

To accomplish the stated objectives in section (3), general conceptual system architecture has been designed to show the main parts of the proposed method as well as the interactions among them.

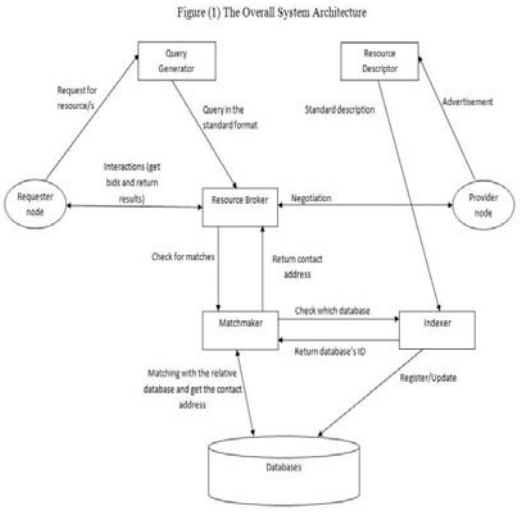


Figure (1) The proposed system architecture

Figure (1) shows the interactions among the system components for two nodes interacting together. As the system subject to this study is a Grid system which consists of N number of providers and M number of requesters, the system components are distributed as follows:

- Resource Broker, Resource Descriptor and Query Generator: are attached with each node.
- Indexer and Matchmaker: there are many Indexers and Matchmakers distributed across the network.

As shown in figure (1), the system gets two inputs: the first input is the resource request from the requester (user or agent); the second input is the resource advertisement from the provider. The system will process the inputs as follows: For the first input (the request), the requester's node will send a request for the required resources to accomplish its job to the query generator. The query generator is the part of the system that is responsible for converting all the requests from the heterogeneous nodes on the open distributed system into a standard unified form[3] which is compatible with the matching method and the advertisement description.

- The output of the query generator is a query in a standard format that will be passed to the main part of the system which is the resource broker.
- The resource broker is responsible for brokering between the requester's node and the provider's node. First of all, the resource broker will ask the matchmaker for matches to check the available advertisements and their nodes contact addresses.
- According to our proposal the system will be open and decentralised so the advertisements will be stored in distributed databases to make the system more immune to any point of failure. Furthermore, the databases will be classified according to the stored resources' functionalities. According to that, the matchmaker will ask the indexer about the appropriate database index. Using this idea, the matchmaker will apply the matching algorithm partially for a subset of the databases rather than all the databases which will reduce the required time to answer the query.
- After that, the matchmaker will return the contact address of the relevant nodes, with advertisements semantically close to the query, to the resource broker.
- The next step is the negotiation where the resource broker will establish the connection between the requester and the available providers to find the best deal for providing the required resources.

For the second input: the resource advertisement from the provider,

- The provider sends the resource specifications to the resource descriptor. Thereafter, the resource descriptor will form a standard description for resource reflecting its functionalities. This step will improve the registering process and accordingly the matchmaking.
- The standard resource description will be passed to the indexer which is responsible, in this case, of storing the resource description in the relevant database according to its functionality specified by the description.

5. The Suggested Implementation Scenarios

According to the proposal, the system is an open multi agent distributed system; after investigation there are two approaches to implement the system based on the architecture shown in figure (1), these scenarios are:

The first scenario is to build the network from scratch; in this scenario, the system's simulation will be built using a network simulator like Opnet[4] or NS2[5]. Accordingly, using this scenario, many issues should be taken in consideration like:

- What type of application will be implemented?

Usually, Grid computing uses for applications that need intensive computing power which are usually scientific research applications like (High Energy Physics HEP, Disaster's Predictions, etc.) where there is a need to process a huge amount of data that would scale to terabytes. Furthermore, there are some commercial applications which need a high computational power like (IPTV: Internet Protocol Television, Peer to Peer mobile resource sharing, etc.) can make use of Grid computing.

- The type of network to be used (Wire or Wireless) and which technology to be used (ex. Wi-Fi, WiMAX, etc.).
- The measures that should be used to assess the fidelity of the implemented methods or algorithms as well as the factors that affect the system performance. In this case there are many factors like: delay, bandwidth, packet size, network topology and messaging system, etc.

The second scenario is to use the Internet/World Wide Web (WWW) as an infrastructure for Grid networking. In this case, any node connected to the Internet can participate in the resource sharing and

[3] For example by using a standard message format like SOAP message format.

[4] See http://www.opnet.com

[5] See http://www.isi.edu/nsnam/ns

requesting process. As one of the most important issues to be considered in open distributed systems is the message passing management, using this scenario we can make use of platform independent message passing protocols like Simple Object Access Protocol (SOAP). As SOAP messages have much overhead information which might affect the performance of the overall performance in term of time, one of our contribution will be the improvement of SOAP messages in a way that makes the SAOP message passing faster and more efficient, keeping the properties of SOAP messages unchanged.

In this scenario, the resources (Hardware, Databases, Programs, Humans, etc.) are described by the services they provide (i.e. describe the resources as services). Service interactions will be facilitated by messages exchange across the system.

By employing multi agent systems techniques, the service providers and requesters will be fully autonomous; service description and implementation will be independent from one node to another. Since the main purpose is to share resources (services in this case), and technically the providers and requesters are sharing information about the services not the services themselves, clear and effective negotiation and contracting policies are required to insure a meaningful information exchange.

To sum up, as described above, in the first scenario, the research work will be more in the direction of network performance with most of the work related to the lower layers of networking like (network and data link layers). In this case, the research will be more application dependent with the intention to improve the network's performance (i.e. developing methods and algorithms to enhance the application itself in order to improve the network performance). This scenario is not suited with our proposal where is has nothing to do with semantic discovery of resources across an open distributed system.

The second scenario, the use of Internet as an infrastructure and the facilities of WWW for interactions among nodes (providers and requesters) is more suitable for our proposal. As described before, the nodes will share information about the resources or services rather than the services themselves so it is more realistic to use semantic techniques in service discovery and information exchange. Furthermore, we can achieve nodes independency, negotiation and contracting using multi agent system's techniques.

6. PROPOSED CONTRBUTIONS

The first contribution will be the system architecture itself, as shown in figure (1); the system will be engineered as a completely decentralised system. First of all, no node has a full control of the system (i.e. each node is responsible for its own control), using this architecture the system will overcome the problem of single point of failure. Furthermore, the advertisements will be stored in many databases distributed across many nodes.

Secondly, the Indexer will classify the advertisements yet the queries according to the resource functionalities across distributed repositories. Accordingly, this technique will facilitate and speed up the job of Matchmaker by looking for matches just in a part of the repository rather than searching all the databases.

Furthermore, the system will not use a software toolkit to gather the resources' information because it might make the system highly dependent on this toolkit. The provider agent will infer the node resources itself and send it to the resource descriptor in a standard platform independent message format associated with the required information for the classification process. Using this standard message format will facilitate the process of interaction among the system components whatever kind of networking is used.

The main contribution will be in the matchmaking process. As the main trade-off in any resource discovery system is between the time and accuracy, our main contribution will be to modulate the search and matching methods in a way to increase the accuracy and decrease the time. The system will employ semantic (linguistic and contextual) features of the resource description as well as non-semantic features. Employing these features is a time consuming process but it will enhance the quality of the matching results. In our research, the main idea is to use complex matching methods (semantic and non-semantic) but at the same time maintaining the time consuming by: first apply the search and matching algorithm for sub-part of the database and secondly by pre-processing the request in a way that makes it more compatible with the advertisements which leads to more accurate result in less time.

Another case to be studied more during our research time is the case where no match is found (i.e. after applying the matching algorithm the result is no match). In this case, the Broker will start to negotiate with the requester agent in order to reduce its requirements according to the available resources.

7. CONCLUSIONS AND FUTURE WORKS

In this paper, a study of resource discovery in distributed environment is presented. We have suggested a decentralised architecture for resource management to improve the overall discovery performance. Two implementation techniques have been studied (building a separated network or relay on the Internet infrastructure and make use of the available WWW tools). From the investigation of these two scenarios we recommend the second scenario which is standard and more compatible with multi agent techniques. The next step in this research is test the fidelity of the suggested architecture as well as the advantages and disadvantages of using multi agent techniques to solve the distributed systems problems.

8. REFERENCES

[1] Abbas, A. (2004). Grid Computing: a Practical Guide to Technology and Applications. Charles River Media.

[2] Amarnath, B., Somasundaram, T., Ellappan, M. And Buyya R. (2009). Ontology-based Grid Resource Management. Software, Practice and

Experience (39), pp. 1419-1438.

[3] Bote-Lorenzo, M., Yannis, A., and G´omez-S´anchez, E.(2004). Grid Characteristics and Uses: A Grid Definition Across Grids, Lecture Notes in Computer Science 2970, pp. 291–298.

[4] Brooke, J., Fellows, D., Garwood, K., and Goble, C. (2004). Semantic matching of Grid Resource Descriptions. In 2nd European Across-Grids Conference. Lecture Notes in Computer Science 3165, pp.240-249.

[5] Castano, S., Ferrara, A. and Montanelli, S. (2005). H-MATCH: an Algorithm for Dynamically Matching Ontologies in Peer-based Systems. in proceedings of the 1st International Workshop on Semantic Web and Databases, pp. 231-250.

[6] Castano, S., Ferrara, A. and Montanelli, S. (2006). Ontology-based Interoperability Services for Semantic Collaboration in Open Networked systems. Springer London, book chapter, pp.135-146.

[7] Elmroth, E., and Tordsson, J. (2008). Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. Future Generation Computer Systems 24(2008): 585–593.

[8] Erdil, D. And Lewis, M.(2007). Grid Resource Scheduling with Gossiping Protocols, In the proceeding of the 7th IEEE international conference on peer to peer Computing, Galway, Ireland, pp. 193-202.

[9] Fasli, M. (2007). Agent Technology for e-Commerce. John Wiley & sons Inc.

[10] Fattahi, S. and Charkari, N. (2009). Distributed ACS Algorithm for Resource Discovery in Grid. International Conference on IT to Celebrate S. Charmonman's 72nd Birthday, March 2009, Thailand.

[11] Freeman,T., Keahey, K.,, Foster1, I., Rana, A., Sotomoayor1, B., and Wuerthwein, F. (2006) . Tools for Growing and Scaling Grids. Lecture Notes in Computer Science 4294,pp.40-51.

[12] Gonble C. and De Roure D. (2007). The Semantic Grid: Myth Busting and Bridge Building. In proceedings of the 16th European Conferece on Artificial Intelligence, ECAI-2004, pp. 1129–1135.

[13] Guan, T., Zaluska, E. and De Roure, D. (2009). An Autonomic Service Discovery Mechanism to Support Pervasive Devices Accessing the Semantic Grid, International Journal of Autonomic Computing, pp. 34-49.

[14] Han, L. and Berry D. (2008). Semantic Supported and Agent Based Decentralised Grid Resource Discovery, Future Generation Computer Systems, pp. 806-812.

[15] Lenica, A., Ogel, F., Peshanski, F., and Briot, J.-P. (2006). Agent-based Grid resource management. In the 2006 International Conference on Computational Science (ICCS'2006), Reading, Royaume-Uni-University, mai.

[16] Li, J. And Vuong, S. (2006). Grid Resource Discovery Based on Semantic P2P Communities. In Proceedings of ACM Symposium in Applied Computing, SAC2006, pp. 754–758.

[17] Li, J. (2010). Grid Resource Discovery Based on Semantically Linked Virtual Organisations, Future Generation Computer systems 26(2006), pp. 361-373.

[18] Ludwig, S. and Reyhan,S. (2006). Semantic Approach to Service Discovery in a Grid Environment, journal of web semantics: science, services and agents on the wide world web 4(2006), pp. 1 -13.

[19] Ludwig, S. and Van Santen, P. (2002). A Grid Service Discovery Matchmaker Based on Ontology Description. In proceedings of the Second International EuroWeb2002 Conference, Oxford, UK, pp. 17-18.

[20] Rood, B. and Lewis, M. (2008). Scheduling on the Grid via Multi-State Resource Availability Prediction. In Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing, pp. 126-135.

[21] Schopf, J., Pearlman, L., Miller, N., Kesselman, C., Foster, I., D'Arcy, M., and Chervenak, A.(2006). Monitoring the Grid with the Globus Toolkit MDS4. Journal of Physics: Conference Series 46 (2006), pp. 521–525.

[22] Somasundaram, T. Balachandar1, R.A., Kandasamy, V., Buyya, R., Raman, R., Mohanram, N., and Varun1, S. (2006). Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker. Technical Report, GRIDS-TR-2006- 10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia.

[23] Timm, I. and Pawlaszczyk, D. (2005).Large Scale Multi-agent Simulation on the Grid. In Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) – (1), pp. 334-341.

[24] Tuchinda, A., Thakkar, S. and Gil, Y. and Deelman, E. (2004). ARTEMIS**:** Integrating Scientific Data on the Grid. In Proceedings of the 16th conference on Innovative applications of artifical intelligence, pp. 892-899.

[25] YANG, Y. (2007). A Scalable Semantic-based Resource Discovery Service for Grids. Master of Science Thesis, Swedish Institute of Computer Science.

This volume contains the papers presented at the Student Session of the 12th European Agent Systems Summer School (EASSS) held on 25th of August 2010 at Ecole Nationale Superieure des Mines de Saint-Etienne, France.
The Student Session, organised by students, is designed to encourage student interaction and feedback from the tutors. By providing the students with a conference-like setup, both in the presentation and in the review process, students have the opportunity to prepare their own submission, go through the selection process and present their work to each other and their interests to their fellow students as well as internationally leading experts in the agent field, both from the theoretical and the practical sector.

ISSN 1864-9300