Position Papers of the Federated Conference on
Computer Science and Information Systems pp. 177–184

DOI: 10.15439/2015F17
ACSIS, Vol. 6

Cognitum Ontorion: Knowledge Representation and Reasoning System

Paweł Kapłański
Gdansk University of Technology
Department of Applied Informatics in Management
Faculty of Management and Economics
Gabriela Narutowicza 11/12, 80-233 Gdansk, Poland
Email: pawel.kaplanski@zie.pg.gda.pl

Paweł Weichbroth
Gdansk University of Technology
Department of Applied Informatics in Management
Faculty of Management and Economics
Gabriela Narutowicza 11/12, 80-233 Gdansk, Poland
Email: pawel.weichbroth@zie.pg.gda.pl

***Abstract*—"If knowledge can create problems, it is not through ignorance that we can solve them." (Isaac Asimov). Nevertheless, at any point of human activity, knowledge (besides practice) is a key factor in understanding and solving any given problem. Nowadays, computer systems have the ability to support their users in an efficient and reliable way. In this paper we present and describe the functionality of the Cognitum Ontorion system. Firstly, we identify emerging issues focused on knowledge representation and reasoning. Secondly, we briefly discuss models and methodology of agent-oriented analysis and design. Next, the semantic knowledge management framework of the system is reviewed. Finally, the usability of Ontorion is argued based on a case study, in which a software process simulation modeling environment is developed. At the end we provide future work directions and final conclusions.**

I. Introduction

DAVID GARVIN notes that "to move ahead, one must often first look behind" [1]. Let us ask this cinch question: how do you want to understand the past, sense the present and predict the future, if you are unable to preserve your knowledge? Indeed, a lot of human effort and material resources have been exploited in preserving knowledge. Indubitably, humans have been using many different forms to express and share knowledge (e.g. drawings, symbols, words and numbers, which nowadays are commonly encoded in computer memory). Now, let us focus on some formal representation methods of knowledge which are the prominent subfield of artificial intelligence (AI).

In the AI canon, knowledge seems to be always defined in a strictly functional way. However, from all incoming questions to the mind of an attentive reader, which one would be the first to reveal the question: What is knowledge? – This might be the question for the majority of us. Bearing in mind a computer "brain" – central processing unit (CPU) is principally able to process sequences of bits where a single bit is represented by two exclusive numbers: 0 (zero) or 1 (one), as a consequence, at the moment we are able to represent "only" facts and rules in computer memory. A distinct fact (or a set of facts), represented by a sentence (or a set of sentences), is used in deductive reasoning. A single rule (or a set of rules), expressed in a form: $if \rightarrow then$, may be a logic or be inductive in its genesis. Elements of particular knowledge (intra- or inter-connected facts or rules) are often named as "knowledge chunks" [2]. A single chunk is commonly attached to an exclusive agent (an independent and separate application unit).

In the beginning, AI research investigated how a single agent can exhibit singular and internal intelligence. However, in recent years, we have observed an interest in concurrency and distribution in AI which have been named as distribution artificial intelligence (DAI). This recent discipline can be divided into two primary areas: distributed problem solving (DPS) and Multi-Agent (MA) systems. It is not a straightforward task to coordinate knowledge, goals and actions among a collection of autonomous agents.

Some successful application of agent-oriented architecture can be pointed in decision support systems (DSS) in the area of the discovery of stock market gamblers patterns [3], [4], web usage mining [5] and the evaluation of information technology [6].

II. Reasoning in agent-oriented design and analysis. A hybrid approach

There is a long history of symbolic reasoning usage in order to provide intelligent behavior in Multi-Agent & Simulation systems (MASS). Deductive Reasoning Agents, which use logic to encode a theory defining the best action to perform in a given situation, are the "purest" in terms of their formal specification. Unfortunately, they suffer from all the practical limitations of formal representation: firstly, the complexity of theorem proofs (it may even lead to undecidable statements) and secondly, the boundaries of expressivity formed by core knowledge representation attributes (e.g. monotonicity of knowledge, open world assumption).

Making deductive reasoning requires the selection of underlying logics that support the nature of agents. It is worth mentioning that the most prominent implementations of deductive reasoning agents are based on intentional logics like formal models of intention logics [7] (e.g. Belief – Desire – Intention, BDI), which take into account some subset of the Saul Kripke modal logic [8].

Problems with symbolic reasoning led to the establishment of the "reactive agent movement" in 1985, revealing an era of reactive agent architecture. The reactive agent movement

manifested in the form of requirements for so-called behavior languages [9]:

1. Intelligent behavior can be generated without explicit representations of the kind that symbolic AI proposes.
2. Intelligent behavior can be generated without explicit abstract reasoning of the kind that symbolic AI proposes.
3. Intelligence is an emergent property of certain complex systems.

Reactive agents are nowadays well recognized but still they lack formal foundations and therefore these kind of MASS are very hard to analyze with formal methods and tools. Nevertheless, the reactive agent movement resulted in Agent Oriented Programming (AOP), e.g. JADE [10], which is currently considered as a step beyond Object Oriented Programming (OOP) in Software Engineering.

A novel approach to designing MA systems – Hybrid Agent Architecture, attempts to combine the best of symbolic and reactive architectures. The system itself is built up of at least two subsystems: (1) a symbolic world model that allows plans to be developed and decisions made and (2) a reactive engine which is capable of reacting to events without involving complex reasoning.

As an example let us consider Ferguson's "TouringMachine" [11], which fits into the definition given above. Ferguson defines: *"The TouringMachine agent architecture comprises three separate control layers: a reactive layer, a planning layer, and a modelling layer. The three layers are concurrently-operating, independently-motivated, and activity-producing: not only is each one independently connected to the agent's sensory apparatus and has its own internal computational mechanisms for processing appropriate aspects of the received perceptual information, but they are also individually connected to the agent's effectory apparatus to which they send, when required, appropriate motor-control and communicative action commands"*.

We present a novel approach to hybrid agent architecture, which is implemented on top of a scalable Knowledge Representation & Reasoning (KRR) system. KRR allows each environment to be described formally as well as giving the possibility to build a reactive agent system based on a knowledge base (KB) triggering subsystem. Moreover, it provides agents with synthesis tools.

Here, we consider a reactive agent that is able to maintain its state [12]. This agent has an internal stand-alone data structure, which is typically used to record information about the state and history of the environment and to store a set of all the internal states of an agent (Fig. 1).

The perception of an agent is realized in its see function if the function is time-independent. The agent's action selection is defined as a mapping from its internal states to actions. The $next$ function maps an internal state and percept to an internal state. The abstract agent control loop is then:

1. Start with the initial internal state $s \leftarrow s_0$.
2. Observe the environment state e, and generate a percept $p \leftarrow see(e)$.

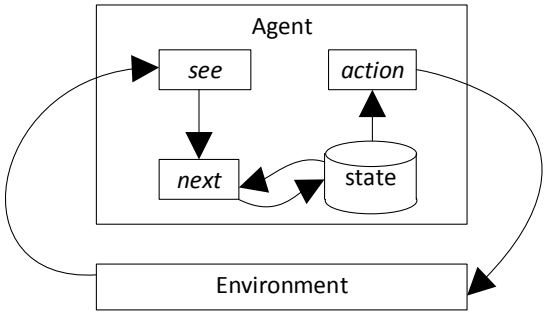

Fig. 1. An agent with its internal state

3. Update the internal state via the $next$ function $s \leftarrow next(s, p)$.
4. Select an action via the $action$ function $a \leftarrow action(s)$.
5. GOTO 2.

The environment state e is (in hybrid architecture) given by the symbolic system – here we use ontology to encode it. The function $next$ is one that needs to be implemented either by the programmer or by an automated process. In the first case, however, it is hard to distinguish such an agent from a (considerably complex) object oriented program (formerly, active-object design pattern implementation [13]). On the other hand, automated agent synthesis is an automatic programming task: given an environment, let's try to automatically generate an agent that succeeds there. The synthesis algorithm should be both sound and complete. Sound means here that the agent will succeed in the given environment once it is correctly constructed, and completeness guarantees the possibility to create the agent for the given environment.

The hybrid approach allows us to build a semi-formal foundation for MASS that allows for a sound and complete synthesis of agents as long as their definition fits into the expressivity frame of underlying logic and if the underlying logic has the reasoning task to be sound and complete itself. This is true for Description Logic (DL) [14] – the foundation for OWL [15], therefore we selected OWL compliant KRR.

III. Ontorion Architecture

Modern Scalable Knowledge Management Systems give the possibility to use KRR in a similar way as we tend to use RDBMS. We have focused on a KRR system the functionality of which allows a user-interface in natural language to be implemented and used.

Ontorion [16] is a Distributed Knowledge Management System that allows semi-natural language to be used to specify and query the knowledge base. It also has a built-in engine trigger which fires the rules each time if the corresponding knowledge is modified. Ontorion supports the major W3C Semantic Web standards: OWL2, SWRL, RDF, SPARQL. Ontologies can easily be imported from various formats, exported to various formats, and accessed with SPARQL [17]. Solutions built on

top of Ontorion can be hosted both in the Cloud and On-Premise environments.

By design, Ontorion allows one to build large, scalable solutions for Semantic Web. The scalability is realized by both – the noSQL, symmetric database Cassandra [18] and the internal ontology modularization algorithm [19]. Ontorion is a cluster of symmetric Nodes, able to perform reasoning on large ontologies. Every single system node is able to do the same operations simultaneously on data sets – it tries to get the minimal suitable ontology module (part) and perform any requested task on it.

The symmetry of the architecture of the cluster provides system scalability and flexibility – Ontorion can be deployed and executed in a computing cloud environment, where the total number of nodes can be changed on request, depending on user requirements.

The fundamental algorithm in a KRR system such as Ontorion ought to reason over description logic selected as a foundation for OWL called $\mathcal{SROIQ}^{(\mathcal{D})}$ [20], and should be able to process complete or selected segments of ontologies.

If performance is more important than expressivity power, then it is possible to switch Ontorion into OWL-RL+ mode. OWL-RL+ mode is constructed in a similar way to how it was first implemented in DLEJena [21]. The reasoning process in OWL-RL+ mode remains in $\mathcal{SROIQ}^{(\mathcal{D})}$ for the T-Box, while for the A-Box the reasoning is based on the OWL-RL ruleset.

Furthermore, the modular separation of complex ontologies also allows the reasoning process to be partitioned, which can be performed on knowledge modules (independent pieces of knowledge) – in parallel, at the same time on separate machines. In other words, the modularization algorithm is scalable and traceable.

In Ontorion, conclusions that are the results of new incoming knowledge can fire triggers at extension/reactive points (Fig. 2). On the other hand, if some chunk of knowledge meets a set of predefined conditions, a knowledge modification trigger executes the procedures responsible for interaction with external systems (e.g. sending a notification using an SMTP server). We observed that knowledge modification triggers allow the Hybrid Agent MASS to be built on top of the Ontorion KRR.

The underlying storage for Ontorion is the BigTable [22] implementation (namely Cassandra), which is able to maintain a petabyte of data. Together with an analytic cluster e.g. Hadoop [23] it forms a BigData solution. In these terms, we can consider Ontorion as a BigKnowledge solution and our Agent application as a BigAgent, which together constitutes a highly scalable hybrid agent infrastructure.

Distributed systems lack the one common model of time. In a distributed environment, time is relative and the serialization of events requires an internode negotiation algorithm. In modern distributed noSQL databases like Cassandra 2.0, the serial time model can be preserved with the use of the Paxos algorithm [24], which allows a distributed atomic "Compare and Set" (CAS) functionally to be efficiently implemented.

CAS in Ontorion is used to maintain the Agent state in a coherent way, therefore the existence of CAS is critical for proper system functioning. CAS also provides a way to make the Agent System fault-tolerant by transactional-queue implementation, which is crucial for long-running simulations.

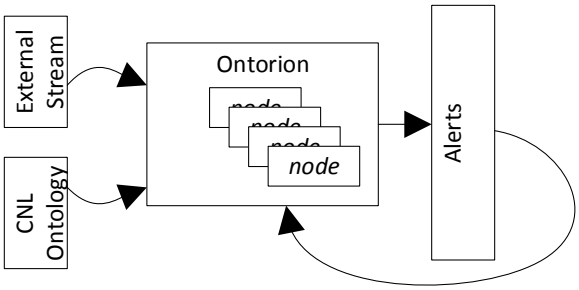

Fig. 2. The Ontorion KRR

IV. Programming Agents with Natural Language

An important, novel feature of Ontorion, among other KRR systems, is the ability to describe knowledge and interact with the user in semi-natural language. The language represents the family of controlled natural languages that is expressive enough to describe OWL. Controlled natural language (CNL) is a subset of natural language with a reduced grammar and vocabulary, which – in this case – translates directly to logic with formal semantics capabilities.

In general, controlled natural language should be unambiguous and intuitive, ultimately forming an easy way for human-machine interaction (understandable by humans, executable by machines). Due to its limitations, it needs to be supported by a predictive (structural) editor which is Ontorion FluentEditor tool [25]. The other, well-known implementation of a controlled natural language is "Attempto Controlled English (ACE)" [26], developed by the University of Zurich. However, the origins of CNL can be found in the famous novel by George Orwell: "1984", where he discusses the NEWSPEAK – a controlled language. The most used industrial implementations nowadays are Domain Specific Language (DSL) (implemented as a part of the Drools project) [27] and Semantics of Business Vocabulary and Rules (SBVR) [28], whereas CNL allows the representation of BPML diagrams.

In Ontorion Fluent Editor controlled language is equipped with formal semantics expressed in logic. General groups of sentences are allowed which include:

1. Concept subsumption, represents all cases where there is a need to specify (or constrain) the fact about a specific concept or instance (or expressions that evaluate the concept or instance) in the form of subsumption (e.g.: *Every cat is a mammal*, *Pawel has two legs* or *One cat that is a brown-one has red eyes*).

2. Role (possibly complex) inclusion specifies the properties and relationships between roles in terms of the expressiveness of $\mathcal{SROIQ}^{(\mathcal{D})}$ (e.g.: *If X loves something that covers Y then X loves-cover-of Y*).
3. Complex rules; If [body] then [head] expressions that are restricted to the DL-Safe SWRL subset [29] of rules (e.g.: *If a scrum-master is-mapped-to a provider and the scrum-master has-streamlining-assessment-processes-sprints-level equal-to 2 then the provider has-service-delivery-level equal-to 1 and the provider has-support-services-level equal-to 2*).
4. Complex OWL expressions; the grammar allows the use of parentheses that can be nested if needed in the form of (that) e.g.: *Every human is something (that is a man or a woman or a hermaphrodite).*
5. Aforementioned knowledge modification triggers that have the form of: *If P then for-each P execute Q*, where P is a premise and Q a consequence. Premise P is an expression that evaluates a set of connected instances that fulfill some conditions, while the consequence Q is a procedure written in $C\#$ programming language (e.g.: fig. 5).

V. The definition of the Multi-Agent system in terms of Knowledge Management System triggers

Here, we present a modern scalable KRR as a foundation for MASS. The discussed KRR (Ontorion) enables the specification of knowledge-modification triggers in the form of reactive rules: if $\rightarrow$ action. Ontorion knowledge modification triggers allow the knowledge itself to be modified and therefore it is possible to build a set of triggers here that are fired continuously. A reactive trigger like this breaks the decidability of the underlying knowledge base and, as a consequence, KRR tasks based on Ontorion are decidable only if all deductive rules are DL-Safe (e.g. they are SWRL equivalent) – otherwise these tasks are non-decidable.

The above property of system modification triggers is very useful for the development of the hybrid MASS. The hybrid agent paradigm can be adapted, by using triggers, even if the environment is modelled in Ontorion as an OWL Ontology, with all its limitations (e.g.: lack of modality or time representation). We can define agents here as OWL individuals. The behaviour of the agents is implemented in reactive rules called $moves$. These rules combine the see, $next$ and $action$ functions discussed earlier (the reactive, state based and abstract model of an agent). Moreover, agent-individuals and "ordinary" OWL individuals are different as agent-individuals are equipped with a transactional, CAS protected internal state, represented by related data-values.

A single $move$ function as a parameter takes a percept which is a result of a reasoning process (here, we consider reasoning as an implementation of the abstract see function) over the current state of the environment. In the implementation of this function a CAS operation is used to preserve transactional semantics. The $move$ function is only activated if the perceived-message is equal to the expected-message.

```
If ... then ... execute <?
    Move(agent, "current-state", message,
        "expected-message", ()=>
    {
         /* the agent action */
        return "new-state";
    });
?>.
```

Fig. 3. General form of the $move$ function

There is not one single agent that activates on perceived-message - it can be any agent that fulfills the rule premise, therefore a rule conclusion can be reused by many agents and the overall execution result of the system is non-deterministic. Messages are transferred between agents using a distributed message queuing system, managed by the KRR.

A single agent is determined by its state and all the $move$ functions that can be ever executed in the context of its state, therefore here, the agent synthesis process, is a process of the assignment of the $move$ functions to the single agent-individual.

The reactive-rule bodies of the $move$ function determine the specific environment state that allows the system to assign the function to the agent; however, the overall behaviour of MASS is non-deterministic. This is due to the fact that the concrete run (the MASS run) needs the selection of agent instances made in runtime – and runtime (in opposition to reasoning time) is a part of the reactive model that is non-deterministic by nature. The non-deterministic selection of choices, often by use of pseudo-random number generators, and the parallel execution of different threads, are required by underlying technologies to provide an efficient computational model.

Therefore, we need to keep in mind, that simulations based on the reactive/hybrid approach are non-deterministic. In this case, we have to perform a large set of experiments with the same initial state and then use analytical tools and methods to verify the statistical hypothesis.

VI. The scalability of KRR oriented MASS

In practical scenarios, when it comes to simulating large societies, it is important to simulate a large amount of agents at the same time. From the technological perspective, currently, we can model large societies of people. Nowadays, the existence of 7×10^9 beings can be encoded in less than 1 GB of memory. If we encode a single human being as a 1 kB vector of bits then we can store an entire population in a single modern hard drive at a relatively low cost. Working with cloud-based environments, we can hire thousands of computers for a few hours with a similar amount of money. Therefore MASS scalability – the ability of the system to scale together with the

size of the problem - is regarded as a mandatory and critical property.

Ontorion is a scalable KRR system, approximately associated with the size of maintained knowledge due to modularization algorithms embedded, whereas Cassandra, as the underlying storage solution, is scalable by its design.

In distributed systems, task synchronization is a burden and sometimes even an obstacle. Task distribution over a set of physical machines demands synchronization protocols. Satisfyingly, the Cassandra database has the Paxos protocol implemented, which allows a global CAS functionality to be implemented. The ability of agents to modify particular chunks of knowledge indicates influence on the surrounding environment as well. What can be seen as a common task in terms of RDBMS (e.g. some database modification), might have large and complex implications in terms of a distributed knowledge base. Given the subset of First Order Logic (FOL), we deal with the monotonic knowledge model. The monotonicity implies that there is no impact on the overall meaning when the order of adding knowledge is one way or another.

When we modify knowledge the problem is somehow more complicated – besides agents tend to modify knowledge very often. The cost of knowledge modification depends on its level, scale and size. The relation between knowledge generality and its modification cost is positive as a result of the replacement of all revalued conclusions. Moreover, knowledge modification triggers, used to implement the *next* functions, break the open world assumption (OWA) [14]. This effect is caused by their ability to modify knowledge depending on the "known" parts of the knowledge. An agent may learn knowledge even when it stays in contradiction to what it already knows. In addition, knowledge modification triggers break the monotonicity of the knowledge base. Therefore, the order of agent Next firing is significant in terms of the final knowledge base shape. As previously mentioned, simulations based on the reactive/hybrid approach are non-deterministic due to both the distributed system properties and the internal non-determinism of the reactive agent system.

VII. Experimental setup - Software Process Simulation Modelling (SPSM) Environment

Software Process Simulation Modelling (SPSM) [30] is widely used nowadays to support planning and control during software development [31], [32]. MA systems play a very important role here as they naturally can be used to simulate social behaviors in the software testing phase. In our approach, the SPSM is divided into two components: ontology and knowledge modification triggers. In the example given below (see Fig. 4), the ontology defines (with CNL) the core concepts such as: competency, task, developer, manager:

We also defined agent-rules by making use of knowledge modification triggers (see Fig. 5,6). Those triggers implement the following scenario: a developer with certain competencies starts to realize a task. After the task is finished, new knowledge about the task realization process is added, and a "Busy"

```
Cpp-Programming is a competency.
Java-Programming is a competency.
...
Task-0 is a task.
Task-1 is a task.
Task-1 is-dependent-on Task-0.
Task-1 requires-competency Cpp-Programming.
Task-1 has-estimated-realization-md equal-to 500.
Task-2 is a task.
Task-2 is-dependent-on Task-1.
Task-2 requires-competency Java-Programming.
Task-2 has-estimated-realization-md equal-to 500.
...
Anna is a developer .
Anna has-competency Cpp-Programming .
Anna has-competency Java-Programming .

John is a developer .
John has-competency Java-Programming .
John has-competency Web-Programming .
...
```

Fig. 4. Configuration of the SPSM environment

state is set on the developer. The second trigger is fired when the task is finished and a "Ready" state is set back.

Every time the environment contains a situation where one task is dependent on the other, finished task, we execute the trigger that forces previous triggers to start a simulation.

Here, we use the *once* function, which ensures that the execution of the trigger happens exactly once (note that this is not a trivial task, not executed in a stand-alone system but in a distributed system which requires dispersed CAS operation). The last step is to define the simulation entry point (see fig. 7).

The start event (see fig. 8) sets up the agents and defines the initial task for the "Done" state to activate the overall simulation.

In the above simulation, to make a long story short, in the beginning, we defined three distinct sets: task, competence and individual. Each agent represented a particular individual (developer). Each task required a precise competency and was time specified. This simplified description of the modelled micro-world was given as an input to the system. Next, on a user request the simulation was executed and a set of rules started to be processed due to accomplish a given set of tasks (Fig. 10).

Ontorion usability has been evidenced in one of many possible applications. In highly complex systems or projects, it seems that it is a considerable issue to design, estimate and finally test all possible dependency relationships between processes and their execution sequence. We showed how to optimize the selection of a developer's competency to a particular tasks. In this instance, we were able to identify "hidden" bottlenecks and constraints.

The experiments performed on the Ontorion cluster show

```
If a task-realization-query requires-competency a
competency and a developer has-competency the
competency and the task-realization-query has-origin
a task then for the task-realization-query and the
developer and the task execute <?
    Move(developer, "Ready",
      task_realization_query, "Programming", ()=>
    {
// read the realization time
var realizationTime =
  (from v in Values where
      v.source==InstanceDL(task) &&
      v.datarole=="have-estimated-realization-md"
   select v.value).
   FirstOrDefault().
   SetConsistencyLevel(ConsistencyLevel.Quorum).
   Execute();

// create the wake-up message
        var msgid = CreateMessage(developer,
"WakeUp",task);

// delayed (by the realization time) modification //
of KB
  KnowledgeInsertWithDelay(
      msgid + " is a wake-up-message."+
      msgid + " has-origin " + developer + "."+
      msgid + " has-task-realization-query "
            + task_realization_query + "."+
      msgid + " has-target "+ developer + ".",
   int.Parse(realizationTime));

// mark the agent state as busy
        return  "Busy";
    });
?>.
```

Fig. 5. The *move* function written in C# is fired when the developer is ready and fit for the given task

```
If a wake-up-message has-target a developer and the
wake-up-message has-origin the developer and the wake-
up-message has-task-realization-query a task-
realization-query and the task-realization-query has-
origin a task then for the wake-up-message and the
developer and the task-realization-query and the task
execute <?
    Move(developer, "Busy",
         wake_up_message, "WakeUp", ()=>
    {
// modify the status of task
       KnowledgeInsert(task+" has-status Done.");

// mark the agent state as ready
       return "Ready";
    });
?>.
```

Fig. 6. The *move* function written in C# is fired when the developer is done with the task

```
If a start-event exists then for the start-event execute
<?
    Once("Lets the simulation start.", ()=>
    {
      CreateAgent("Mark","Ready");
      CreateAgent("John","Ready");
      CreateAgent("Tom","Ready");
      CreateAgent("Gabi","Ready");
      CreateAgent("Anna","Ready");

      KnowledgeInsert("Task-0 has-status Done.");
    });
?>.
```

Fig. 7. The simulation entry point

```
Start-Event is a start-event.
```

Fig. 8. The simulation start event

flexible system scalability, persistent intra-communication duration between nodes and overall system stability. As an illustration, let us present this factual operation. A new node added to the server farm was properly initialized and broadcasted to other nodes and a scheduled job was again distributed. Still, a systematic empirical measurement needs to be made to monitor system behavior, especially when some changes have taken place. A cloud-based environment is our choice due to the obvious benefits of machine virtualization.

For Ontorion cluster setup, we used cluster of 3 standard VM nodes. On top of this cluster we executed MASS made of 5 agents.

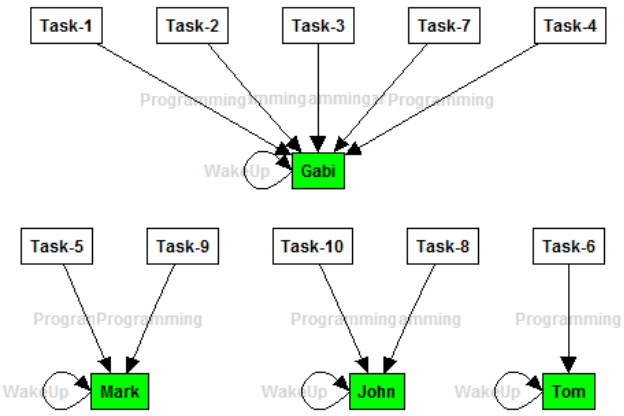

Fig. 9. The result of a particular SPSM simulation (assignment of programmers to tasks)

```
0,000   Anna   Ready   Programming     Busy    Task-1
1,782   Anna   Busy    WakeUp  Ready   Task-1
3,058   Anna   Ready   Programming     Busy    Task-2
4,380   Anna   Busy    WakeUp  Ready   Task-2
10,239  Anna   Ready   Programming     Busy    Task-11
10,332  Gabi   Ready   Programming     Busy    Task-12
10,598  John   Ready   Programming     Busy    Task-7
10,848  Mark   Ready   Programming     Busy    Task-5
12,431  Tom    Ready   Programming     Busy    Task-6
16,593  Mark   Busy    WakeUp  Ready   Task-5
16,749  Mark   Ready   Programming     Busy    Task-4
19,306  Gabi   Busy    WakeUp  Ready   Task-12
20,254  Gabi   Ready   Programming     Busy    Task-9
26,214  Mark   Busy    WakeUp  Ready   Task-4
30,358  Mark   Ready   Programming     Busy    Task-10
31,228  Gabi   Busy    WakeUp  Ready   Task-9
31,579  Tom    Busy    WakeUp  Ready   Task-6
34,636  Gabi   Ready   Programming     Busy    Task-3
36,639  Mark   Busy    WakeUp  Ready   Task-10
39,322  Mark   Ready   Programming     Busy    Task-8
41,086  Anna   Busy    WakeUp  Ready   Task-11
49,549  Gabi   Busy    WakeUp  Ready   Task-3
49,972  Mark   Busy    WakeUp  Ready   Task-8
58,962  John   Busy    WakeUp  Ready   Task-7
END
```

Fig. 10. The system's console with detailed information of task status, performer and dependencies

VIII. Examples of Ontorion applications

Due to the limitations of this paper, we only briefly mark a few Ontorion applications which we think give an objective perspective on its functionality.

First of all, our system has been successfully deployed as an intelligent semantic tool in a company from the energy sector located in the USA. There are several benefits worth mentioning of customizing Ontorion to this client. Primarily, we were able to semantically describe data sources due to automating the process of infrastructure management.

Another interesting application, which has taken place recently in a company in the Aeronautics and Space industry, is a case-based reasoning solution. Here, we combine text mining with a dedicated ontology to mine and structure information residing inside messages, incoming from users, which expose some third-party system errors and defects. Next, to those extracted chunks of information, the inference engine adds relevant tags based on a semantic analysis and together, such enhanced information (someone may even define such rich data as knowledge [33]) is exported to a database (knowledge base). Later, we execute triggers to combine this knowledge with rules and reason a possible set of actions. Nevertheless, an expert is responsible for selecting the final action to be taken due to the solution of the reported problem.

In medicine, for the Maria Skłodowska-Curie Institute of Oncology we built a highly complex ontology which represents a set of rules describing cancer procedure treatment. First and foremost, we are able to centrally manage all the rules, where, on a user's rule change request, with little effort, we just need to modify the semantic description preserving other rules from unnecessary modifications. The most beneficial is the usage of (semi) natural language that is readable for medical oncology experts. Thus in this way they are able to verify the knowledge input.

In the production sector, we have developed a common dictionary for different actors to communicate and collaborate world-wide. Actors include employees (located in different countries and continents) and heterogeneous information systems (from different software vendors). The deployment of semi-natural language, integrated inside the Fluent Editor (Ontorion ontology editor), proved to be a tool, on the one hand, easy to understand and use by its users, and on the other hand, easy to configure and maintain for administrators during systems (applications) integration.

Finally, we can use Ontorion to manage authorization and authentication, versioning, auditing and what distinguishes us from the competition is collaborative ontology engineering. Moreover, we are able to deploy a solution for semantic enhance searching which, based on taxonomy, efficiently improves sharing and searching for information in response to a user query, and interpreting strings of words not only using statistical techniques, but in the sense of logical connections existing between them. On the other hand, such a taxonomy can be seen as an asset of organization knowledge, which can be used to acquire knowledge from individuals and next to preserve it in a formal way.

IX. Future work directions

First, we plan to explore the scalability of MASS created in the way described in this paper. Even if Ontorion itself is a scalable solution, it is not obvious that the knowledge and inter-agent communication via a distributed queue will have the property of scalability too. Secondly, we want to explore an automated agent synthesis, based on theorem proofers. The synthesis should select rules in an adaptive way from the set of available rules by activating them with some threshold.

X. Conclusions

In this paper, we showed that the Ontorion server is able to execute, maintain and control massive simulations based on a hybrid MASS approach. The resulting MASS fits well into the definition of a Hybrid MASS. The perception of an agent is realized by description logic theorem proofers (reasoners). Agents are modelled as instances equipped with a relevant time model, which allows them to interact with the environment and with each other. Inter-agent communication is realized by messages that together with the environment are represented by an ontology managed by the server. Finally, actions performed by agents are encoded in a particular programming language, which brings our approach close to AOP.

An agent synthesis benefits from a formal reasoning engine (being a central component of KRR) and is based on an action-selection procedure. An expressive and distinct Ontorion functionality is the ability to encode agent logics in semi-natural language in the interest of a less professional user. We also observed that this allows end users to understand actions taken

by an agent, even if a user is not well trained in formal representation systems.

Obviously, we are aware of some obstacles which can be pointed out in the presented simulation. The key issue is to verify the usability and then estimate the degree of functionality adoption on the client side. We have also not used any technique of knowledge verification and validation [34]. On the other hand, we presented some Ontorion applications in the medicine, aerospace and production industries which were positively evaluated and are still in use. Based upon preliminary feedback from our clients, we think that the presented system, as a multi-agent simulation platform, is a promising prospect not limited to any particular industry or purpose.

Ontorion is free of charge for academic institutions and independent researches. For more information, please visit our website available at http://www.conitum.eu/semantics/.

REFERENCES

[1] D. Garvin, *Learning in Action: A Guide to Putting the Learning Organization to Work*. Harvard Business School Press, 2000. ISBN 9781578512515. [Online]. Available: http://books.google.co.nz/books?id=HAaZbow_cpUC

[2] J. A. Jakubczyc and M. L. Owoc, "Contextual knowledge granularity," in *Proceedings of Informing Science & IT Education Conference (InSITE)*, 2011, pp. 259–268.

[3] M. Bac, J. Korczak, A. Fafula, and K. Drelczuk, "A-trader - consulting agent platform for stock exchange gamblers." in *FedCSIS*, 2012, pp. 963–968.

[4] J. Korczak, M. Hernes, and M. Bac, "Risk avoiding strategy in multi-agent trading system." in *FedCSIS*, 2013, pp. 1119–1126.

[5] P. Weichbroth and M. Owoc, "A framework for web usage mining based on multi-agent and expert system an application to web server log files," *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, no. 206, pp. 139–151, 2011.

[6] C. Orłowski, A. Ziółkowski, and A. Czarnecki, "Validation of an agent and ontology-based information technology assessment system," *Cybernetics and Systems: An International Journal*, vol. 41, no. 1, pp. 62–74, 2010.

[7] A. S. Rao and M. P. George, "BDI agents: From theory to practice," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, pp. 312–319. [Online]. Available: http://www.agent.ai/doc/upload/200302/rao95.pdf

[8] S. Kripke, *Naming and necessity*. Harvard University Press, 1980. ISBN 9780674598461. [Online]. Available: http://bks0.books.google.com.ec/books?id=9vvAlOBfq0kC

[9] R. A. Brooks, "Intelligence without Reason," in *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, 1991, pp. 569–595.

[10] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - A FIPA-compliant agent framework," CSELT, Tech. Rep., 1999.

[11] I. A. Ferguson, "Touring machines: Autonomous agents with attitudes," *Computer*, vol. 25, no. 5, pp. 51–55, May 1992. doi: 10.1109/2.144395. [Online]. Available: http://dx.doi.org/10.1109/2.144395

[12] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN 047149691X

[13] D. Schmidt, M. Stal, H. Rohnert, and F. Buschman, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, ser. Wiley Series in Software Design Patterns. John Wiley & Sons, 2000, vol. 2.

[14] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, January 2003. ISBN 0521781760

[15] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer," World Wide Web Consortium, W3C Recommendation, October 2009. [Online]. Available: http://www.w3.org/TR/owl2-primer/

[16] Cognitum. Ontorion Semantic Knowledge Management Framework. http://www.cognitum.eu/semantics/ontorion/. Made available on 20 March 2015.

[17] B. Quilitz and U. Leser, "Querying distributed rdf data sources with SPARQL," in *The Semantic Web: Research and Applications*, ser. Lecture Notes in Computer Science, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Springer Berlin Heidelberg, 2008, vol. 5021, pp. 524–538. ISBN 978-3-540-68233-2. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68234-9_39

[18] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/sigops/sigops44.html#LakshmanM10

[19] P. Kaplanski, "Syntactic modular decomposition of large ontologies with relational database," in *ICCCI (SCI Volume)*, 2009, pp. 65–72.

[20] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible sroiq." in *KR*, P. Doherty, J. Mylopoulos, and C. A. Welty, Eds. AAAI Press, 2006. ISBN 978-1-57735-271-6 pp. 57–67.

[21] G. Meditskos and N. Bassiliades, "Dlejena: A practical forward-chaining owl 2 rl reasoner combining jena and pellet," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 1, 2010. [Online]. Available: http://www.websemanticsjournal.org/index.php/ps/article/view/176

[22] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006.

[23] T. White, *Hadoop: The Definitive Guide*, first edition ed., M. Loukides, Ed. O'Reilly, june 2009. [Online]. Available: http://oreilly.com/catalog/9780596521981

[24] L. Lamport, "Paxos made simple, fast, and byzantine," in *OPODIS*, 2002, pp. 7–9.

[25] Cognitum, "Fluent Editor 2014 - Ontology Editor," http://www.cognitum.eu/semantics/FluentEditor/, made available on 20 March 2015.

[26] N. E. Fuchs, U. Schwertel, and R. Schwitter, "Attempto controlled english - not just another logic specification language," in *LOPSTR '98: Proceedings of the 8th International Workshop on Logic Programming Synthesis and Transformation*. London, UK: Springer-Verlag, 1990. ISBN 3-540-65765-7 pp. 1–20.

[27] M. Proctor, M. Neale, B. McWhirter, K. Verlaenen, E. Tirelli, A. Bagerman, M. Frandsen, F. Meyer, G. D. Smet, T. Rikkola, S. Williams, and B. Truit, "Drools," 2007. [Online]. Available: http://labs.jboss.com/drools/

[28] OMG. (2008) Semantics of business vocabulary and business rules (sbvr), v1.0. http://www.omg.org/spec/SBVR/1.0/PDF, Abruf am 02.05.2013. [Online]. Available: http://www.omg.org/spec/SBVR/1.0/PDF

[29] B. Glimm, M. Horridge, B. Parsia, and P. F. Patel-Schneider, "A syntax for rules in OWL 2." in *OWLED*, ser. CEUR Workshop Proceedings, R. Hoekstra and P. F. Patel-Schneider, Eds., vol. 529. CEUR-WS.org, 2008. [Online]. Available: http://dblp.uni-trier.de/db/conf/semweb/owled2009.html#GlimmHPP08

[30] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software process simulation modeling: Why? what," *Journal of Systems and Software*, vol. 46, pp. 91–105, 1999.

[31] A. Czarnecki, C. Orłowski, T. Sitek, and A. Ziółkowski, "Information technology assessment using a functional prototype of the agent based system," *Foundations of Control and Management Sciences*, vol. 9, pp. 7–28, 2008.

[32] A. Czarnecki and C. Orłowski, "Ontology as a tool for the it management standards support," in *Agent and Multi-Agent Systems: Technologies and Applications*, ser. Lecture Notes in Computer Science, P. Jędrzejowicz, N. Nguyen, R. Howlet, and L. Jain, Eds. Springer Berlin Heidelberg, 2010, vol. 6071, pp. 330–339. ISBN 978-3-642-13540-8. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13541-5_34

[33] J. Gołuchowski, "Technologie informatyczne w zarządzaniu wiedzą w organizacji." *Prace Naukowe. Akademia Ekonomiczna w Katowicach*, 2007.

[34] M. A. Mach and M. L. Owoc, "Validation as the integral part of a knowledge management process," in *Proceeding of Informing Science Conference*, 2001.