BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 5, 2018
DOI: 10.24425/124277

Distributed control systems integration and management with an ontology-based multi-agent system

D. CHOIŃSKI* and M. SENIK

Silesian University of Technology, ul. Akademicka 16, 44-100 Gliwice, Poland

Abstract. The purpose of this study is to create, analyze and reuse an ontology-based approach during implementation of a multi-agent system (MAS) capable of integrating different elements of a distributed control system (DCS). Ontology is considered as knowledge about a particular domain. It includes static description of the domain's structure and properties, by means of which it is possible to define the domain's dynamic states, transitions between those states and conditions of those transitions. Because of that, it is possible to analyze such ontology in terms of modal logic in predicate logic settings.

Key words: distributed control system, multi-agent system, software based integration systems, ontology, Petri nets.

1. Introduction

This paper addresses the problem of an efficient approach to distributed control system (DCS) integration engineering. This engineering is based on the knowledge gathered from the integrated system and modeled by means of ontology [1]. The presented approach to DCS integration engineering provides an ontology-based multi-agent solution to a problem of how to develop and maintain complex software-based integration systems (IS) over their lifetime under dynamic and demanding conditions of the market in the various different branches of manufacturing industries. Bringing together configuration and customization of a wide array of different (new or existing, usually incompatible) hardware or software along with third party subsystems which accomplish a single goal of creating a unified, fully functional system according to the customers' requirements and needs, is commonly referred to as DCS integration. One of the major problems that need to be answered during DCS integration is how to format, pass and process the obtained data. This is of significance because DCS integration engineering focuses on the notion of harnessing all available information, originating at a variety of different points, into one complete picture. However acquisition of those different pieces of information cannot be achieved without suitable communication interface. Moreover, it is often the case that the integration is harder to achieve the greater the number of integrated subsystems that are involved.

DCSs are usually involved in some type of production processes and without a proper approach to an integration the time between requirements formalization and the final product release can be unacceptably long. Overcoming those problems requires scalability and openness of the integration system. A well-designed DCS integration system is a future-proof solution characterized by interconnectivity and interoperability that directly impacts the system's openness and scalability. In the proposed approach, an attempt was made at establishing a reasonable solution to the above-described problems by means of introducing modal logic analysis in predicate logic settings of the ontology that is applied in multi-agent DCS integration engineering. It is worth mentioning that multi-agent solutions have a great potential to solve problems that are not directly connected with a given industry, such as formation control [2] or population-based global optimization meta-heuristics [3].

2. Knowledge representation for the distributed control system

DCSs are usually heterogeneous, which means that they involve a great number of different subsystems. Heterogeneity is the compositional relationship between numerous interconnected and interoperable third party subsystems that are usually distributed geographically and constitute a source of different pieces of information. Heterogeneity of a DCS integration system is made possible through standard conforming hardware or software elements. This means that each such element uses well-known, common interface that allows for easy and efficient communication and cooperation. Because of that, it is very important to provide such a DCS integration solution that will enable easy and understandable consolidation of data. From a software point of view, well-designed DCS integration systems achieve that by presenting a loosely coupled structure. Such an approach enforces interface-based implementation. This maximizes integration system efficiency, reliability and safety, extending its lifespan as regards the changing environment [4].

Knowledge management relates to the notion of capturing, developing, sharing and using domain-specific knowledge. This knowledge is enclosed inside the DCS and used to assists the

*e-mail: Dariusz.Choinski@polsl.pl

Manuscript submitted 2017-11-25, initially accepted for publication 2018-02-08, published in October 2018.

human operator in the decision making processes. However, the quality of how this process is performed by different DCSs varies, thus each company is particularly interested in an autonomous, intelligent, intuitive and maintainable integration system characterized by interoperability and interconnectivity that will manage the domain knowledge most effectively. Such a system has to respond to users' requests or environmental changes and take initiatives, attending to users' needs [5, 6]. Similarly to the integration system, domain knowledge is heterogeneous and distributed, as it relates to different system parts. To be meaningful, unambiguous and useful, this knowledge needs to use common terminology. This is achieved by means of ontologies that can be used during integration system design, runtime and maintenance [5, 7].

DCSs present a strong, hierarchical and layered design [8]. Interaction between each layer, depending on the careful selection of communication interfaces and data structures, is a key factor determining the DCS's overall quality and efficiency [9]. Because of that, various different IA vendors of control and instrumentation systems are gradually investing more valuable resources in versatile technological research, and are taking advantage of the latest trends and developments in hardware and software, making their products more intuitive, consistent and easier to configure [9]. To achieve this, each vendor usually focuses on and contributes to standardized solutions rather than designing proprietary, costly and private ones. Having standardized solutions assures simplification and reusability of control system interfacing while increasing the enterprise's long term competitiveness, recognizability and income at the same time [10, 8, 11]. Open platform communications (OPC) is a great example of such an approach, and there are multiple studies concerning practical utilization of OPC, such as [11] and [12], to name but a few.

OPC is an open automation interface protocol actively developed by the OPC Foundation [13, 14]. OPC was established as a method for fast and efficient data retrieval, integration, analysis and communication between different, third party automation devices and heterogeneous subsystems [15, 16]. OPC enables easy and efficient data exchange between OPC compliant hardware and software components developed by different automation vendors. Because OPC specifications can be applied in different fields of automation applications, they are largely independent from one another, however it is a straightforward task to combine them together in a single application. Today, the vast majority of OPC compliant products implements the classic OPC Data Access (DA) interface, which is used for real time data acquisition from remote process controllers [14, 15].

3. Ontology-based interfaces and data structures

Ontology is the study of various different kind-of and part-of hierarchies, relations and categories of elements that exists or may exist. To be of any use, ontologies have to provide some terminology which is commonly referred to as domain vocabulary. Such vocabulary allows referring to different, domain specific ontological concepts, helping to describe each single domain specific element concisely, unambiguously, independently of any reader and context, and in more detail by specifying its hierarchies, constraints and composition rules. It is also very important to stress that ontological vocabulary is always provided in a machine interpretable and platform independent format, thus, if required, it is a straightforward task to translate it between different languages, essentially without any compatibility problems or the need for additional changes in the ontology itself. Based on the obtained vocabulary, a concept hierarchy commonly referred to as a taxonomy can be specified. Formally, taxonomy reuses the software class notion, simply because such an approach allows for applying composition and inheritance mechanisms efficiently. Class notion is both human and machine interpretable, and supports easy, bidirectional conversion into other ontology formats. With ontology and on the basis of common and shared domain characteristics, a taxonomy can be expressed with mutually exclusive, unambiguous clustering and strict generalization of classes. It is worth mentioning that each ontological class corresponds to a single ontological concept. Both are more or less accurate approximations of the same reality. However a concept is only its abstraction whereas a software class is its formal model. In the object-oriented (OO) programming context, class hierarchies and composition can be easily reproduced in each OO programming language such as Java, C# or C++ as well as in UML, XSD and FOL. This allows different engineers with different fields of expertise to choose among different ontological formats to share their knowledge most efficiently and synchronize it with the existing ontology.

4. Petri nets for modal logic ontological MAS analysis

A Petri net is a discrete event dynamic system and a mathematical modeling language allowing for the description of complex, concurrent and distributed systems [17–20]. Its structure C can be described as a tuple of four elements (1) in which P is a finite set of places of cardinality of n, T is a finite set of transitions of cardinality of m, I is an input function and O is an output function.

$$C = (P, T, I, O) \tag{1}$$

A marked Petri net M (2) is referred to as a structure consisting of tuple C and marking μ in Fig. 1. Petri net marking is a function that relates places P to nonnegative integers N.

$$M = (C, \mu) \tag{2}$$

Tokens reflect the dynamic nature of a modeled system and can be assigned to and reside inside Petri net places only. By doing so, they can control the execution of the transitions. Consequently, their number as well as positions can change during the Petri net execution.

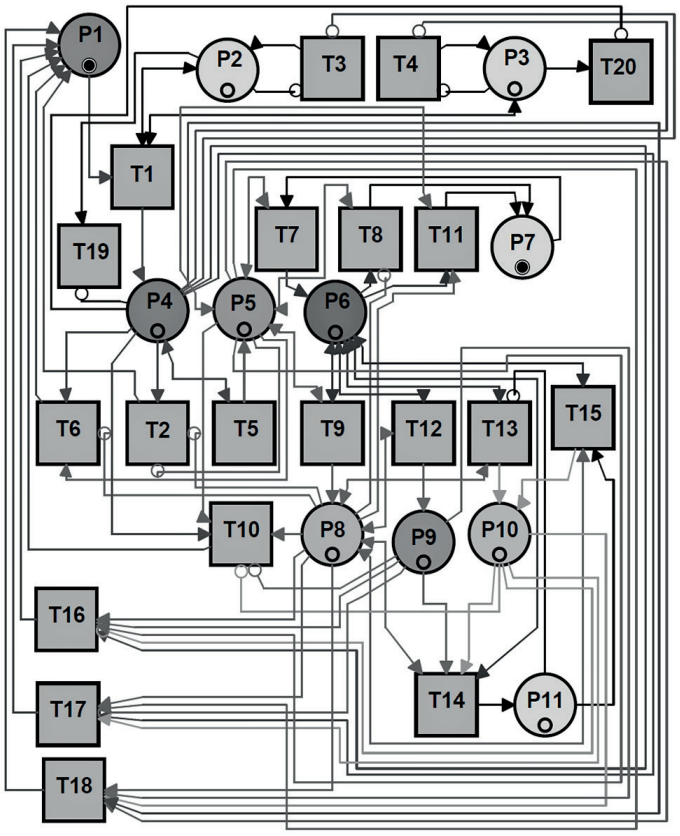

Fig. 1. OPC agent and OPC DB agent interaction – Petri net

Petri net marking μ defines its state. Each time an enabled transition fires a change in a Petri net, a change in its state occurs. This change can be described by a two-argument δ next-state function. The result of the next state function is a new marking, i.e. μ'. Consequently, the sequence of resulting markings $(\mu_0, \mu_1, \mu_2, \ldots)$ and the sequence of fired transitions $(t_{j0}, t_{j1}, t_{j2}, \ldots)$ are related by means of a δ next-state function.

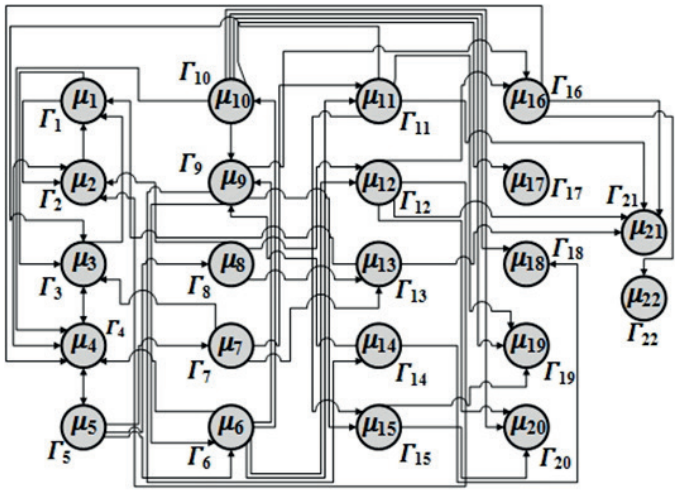

Fig. 2.OPC agent and OPC DB agent interaction – part of a Petri net marking graph, based on the accessibility relation set and Fig. 1

For the purposes of further modal logic analysis, it is important to note that each Petri net marking μ_i is considered a separate world inside which given logical formulas will be examined, as per Fig. 2.

Classical modal logic extends classical propositional and predicate logic and includes additional possibility ◊ and necessity □ operators that express modality.

$$\Diamond P \leftrightarrow \neg\Box\,\neg P \tag{3}$$

$$\Box P \leftrightarrow \neg\Diamond\,\neg P \tag{4}$$

Both operators are unary and can be expressed in terms of the other with negation so that (3) is equivalent to (4). The most often used modal logic system is S5 although additional systems also exist. The S5 system defines the necessity and possibility modalities. Based on S5, a proposition is necessary if it is true in all possible worlds. A proposition is possible if it is true only in some possible worlds. Given the above, it can be stated that there can be many things that are true in various possible worlds. However, if something is true in every world, it is said that it is necessary. On a contrary, if something is possible in some or at least one world, it is said it is possible. Based on the S5 system, it is possible to perform a detailed analysis on a per world basis of any complex system being engineered.

In the presented approach to an engineered system analysis, it is required to determine two non-empty sets that will contain possible worlds W and relations R that exist between the members of a possible worlds set. This situation can be described via the following set of logical equations:

$$\begin{aligned}
&\forall\, \mu_i \,\exists\, W.\ \ \mu_i \in W.\\
&\forall\, R_i \,\exists\, R.\ \ R_i \in R.\\
&\forall\, r_{i:j} \,\exists\, \mu_i, \mu_j.\ \ r_{i:j} \in R_i \wedge r_{i:j} = \{\mu_i, \mu_j\} \wedge \mu_i R_i \mu_j.\\
&\forall\, \mu_i, \mu_j \,\exists\, t_{i:j}.\ \ t_{i:j} \in T \wedge \mu_i R_i \mu_j \Rightarrow \delta(\mu_i, t_{i:j}) = \mu_j.
\end{aligned} \tag{5}$$

Based on (5), it can be said that the state of a world μ_j is a direct possibility for world μ_i i.e.: μ_i R μ_j. The last step of the initial analysis stage is the determination of a set of logical formulas Γ true for each world μ_i in (6), that is:

$$\begin{aligned}
&\forall\, \Gamma_i \,\exists\, \Gamma.\ \ \Gamma_i \in \Gamma.\\
&\forall\, \mu_i \,\exists\, \Gamma_i.\ \ (\mu_i, \Gamma_i) \Rightarrow \mu_i \models \Gamma_i.
\end{aligned} \tag{6}$$

The ν symbol is called a valuation function, which assigns objects from a domain of discourse to each term inside each logical formula of Γ_i. Together, non-empty set of worlds W, their relations R and valuation function ν form a model M of an engineered system (7), such that:

$$M = (W, R, \nu). \tag{7}$$

5. Modal logic based ontology analysis

Propositional logic satisfiability is decidable but not fully capable of addressing all the characteristic properties of complex systems such as MASs. The domain that is to be represented and reasoned about consists of a number of objects with a variety of properties and relations among them. Propositional logic represents only statements about the domain, without reflecting its internal structure and without modeling its entities explicitly. Consequently, in the propositional logic context such domain knowledge is hard to encode. A possible solution is to introduce variables and allow for quantification in logical statements. In such a case, first-order logic (FOL) is the right choice.

FOL eliminates deficiencies of propositional logic by representing objects, their properties, relations and statements. It introduces variables that substitute and refer to arbitrary objects. It also introduces quantifiers which allow for making statements about groups of objects possible without the need to represent each of them separately. However the satisfiability and decidability of the FOL formulas, and thus their validity, cannot be clearly verified.

From an analytical perspective, one useful solution is modal logic [21–23]. Modal logic is an extension of classical propositional logic. However it can also be studied in FOL settings [24, 25]. In modal logic, evaluation of formulae occurs within a fixed set of worlds rather than in a single world. Thus it is possible that a formula may be true in some worlds and false in others. Navigation between different worlds is expressed through the accessibility relation which characterizes the changing nature of each dynamic system. Consequently, it is possible to model entire system in terms of sequence of worlds.

Figure 2 and Table 1 depict this in detail. In modal logic settings, satisfiability and decidability of FOL formulas can be verified on per world basis, without having the problem of enumerating each interpretation [24] because it is possible to find such a world and such interpretation under which this formula is true [26]. Such interpretation is then referred to as a formula model [27]. On this basis, a set of logical formulas under particular interpretation can be used as a logical specification that describes a dynamic system in terms of single worlds or, more precisely, states. Such approach is correct given that evaluation of FOL formulas is based on modal logic and focuses on a certain set of worlds. This reflects the finite state characteristics of complex systems such as MASs.

In modal logic terms, a model of a logical formula is referred to as an ordered triple (7) that consists of a non-empty set of possible worlds W, a binary accessibility relation R that holds (or not) between the possible worlds and a valuation function ν that, under a given interpretation, binds domain-specific objects with each term inside each logical formula ensuring formula satisfiability in each possible world. The accessibility relation means that the state of affairs of one world is reachable or possible for another world.

Such an approach allows to formally verify and check correctness of an abstract mathematical model of a real system in terms of its algorithms and properties with respect to its specification. This formal verification usually explores all worlds

Table 1
OPC agent and OPC DB agent interaction – part of possible marking set μ based on Fig. 1 and Fig. 2

	Places										
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
μ_1	1	0	0	0	0	0	1	0	0	0	0
μ_2	1	1	0	0	0	0	1	0	0	0	0
μ_3	1	0	1	0	0	0	1	0	0	0	0
μ_4	1	1	1	0	0	0	1	0	0	0	0
μ_5	0	1	1	1	0	0	1	0	0	0	0
μ_6	0	1	1	1	1	0	1	0	0	0	0
μ_7	0	1	1	1	1	1	0	0	0	0	0
μ_8	1	1	1	0	0	1	0	0	0	0	0
μ_9	0	1	1	1	1	1	0	1	0	0	0
μ_{10}	0	1	1	1	0	1	0	0	0	0	0
μ_{11}	1	0	1	0	0	1	0	0	0	0	0
μ_{12}	1	1	0	0	0	1	0	0	0	0	0
μ_{13}	0	1	1	1	1	1	0	1	1	0	0
μ_{14}	0	1	1	1	1	1	0	1	0	1	0
μ_{15}	1	0	0	0	0	1	0	0	0	0	0
μ_{16}	0	1	1	1	1	0	1	0	1	0	0

(states) and relations (transitions) that exists within the model. It is worth mentioning that abstract mathematical models are usually expressed via finite state machines, transition systems, Petri nets, timed and hybrid automata, process algebra and formal semantics of programming languages. Model checking is one of the most successful formal verification techniques. In this approach, the verified system S can be represented by a logical modal [27] model M while the verified statement P is expressed by a logical formula Γ_P. Consequently, verification via model checking [27, 28] is an activity that strives to provide an answer on whether logical formula Γ_P satisfies the logical modal model M (8).

$$M \models \Gamma_P \tag{8}$$

In the modal logic model, checking the decision problem considers whether a logical formula, which forms part of specification of a particular system, is satisfiable in the model that represents each possible evolution of this system [24, 29].

In the presented approach, a practical study of an ontology-based DCS MAS integration system in the finite states setting is introduced. The result is a manual, analytic tool that allows to formally verify properties, algorithms and structure of a designed ontology under modal logic settings. During the initial design stage, ontology is created by means of UML class diagrams. When ready, UML-based ontology needs to be verified to check whether it captures all the desired DCS system features correctly without any serious inconsistencies in UML

classes' properties, composition and hierarchy. This is because usually UML class diagrams tend to hide many such problems until they are discovered during implementation or tests. Consequently, this can seriously impact both functionality and quality of the solution being delivered. To avoid such situation, careful analysis is required. In many recent studies first-order logic (FOL) is recognized as a suitable analytical tool that has sufficient power to perform successful UML class diagram verification [30–32]. However, in general, FOL satisfiability is not decidable, thus an UML-based FOL model cannot be clearly verified. In modal logic, satisfiability and decidability of UML-based FOL formulas can be achieved on a model of modal logic worlds. The model represents all possible evolutions of an underlying system and consists of a set of possible worlds (states), relations (transitions) between those worlds and associated logical formulas (specifications). Consequently, the original problem of ontology verification becomes a problem of formal verification of a model of UML-based FOL formulas under modal logic settings.

When ready, ontology will be used during implementation and eventually during runtime of a MAS-based DCS integration system. Each MAS-based system is composed of different types of agents that perform different operations on behalf of other agents during runtime. By achieving their smaller individual goals the agents achieve greater goals of their parent system together. At the design stage, each agent can be described by means of a Petri net with which it is possible to differentiate each single state in which the agent can reside, transitions between those states and conditions of those transitions. By doing so, it is possible to indirectly obtain a complete set of worlds (states) and their relations (transitions) for the formal verification of an ontology or, more precisely, for the formal verification of a model of UML-based FOL formulas under modal logic settings. In each single world it is possible to use an effective method to find such an interpretation or interpretations that will evaluate the truth of each FOL-based formula. Such an approach is correct because each agent and its ontology evolves together during runtime. Thus an agent state becomes an ontology state as well [29, 33]. With that, it is possible to put into motion the idea of formal ontology-based UML model verification using model checking under modal logic settings as each required piece of a modal logic model (i.e. set of worlds, their relations and interpretation function) can be easily determined. Such an approach to formal ontology-based UML model verification allows to capture not only structural inconsistencies in the UML class diagram properties, composition or hierarchy but functional inconsistencies of each agent type that forms the MAS-based DCS integration system before its implementation. It is worth mentioning that each agent type has different responsibilities inside its parent MAS. Each such agent has a different Petri net state graph or graphs and operates as part of an ontology. Therefore the entire process of formal ontology-based UML model verification will be divided between different types of agents. Consequently, in terms of a single agent type, formal ontology-based UML model verification will be executed using a smaller domain of discourse.

6. Sample analysis

Based on the example, it can be said that the net $C = (P, T, I, O)$ consists of a set of places $P = \{p_1, p_2, \ldots, p_{11}\}$, with exactly $\#(P) = 11$ elements, as listed in Table 2. Places P are connected with transitions $T = \{t_1, t_2, \ldots, t_{20}\}$ that consist of $\#(T) = 20$ elements, as per Table 3. Input function I and output function

Table 2
OPC agent and OPC DB agent interaction – Petri net places

Place	Description
P1	OA initial inactive state
P2	OPC server available
P3	OPC server namespace available
P4	OA initialized
P5	DBA initialized
P6	DB reachable
P7	DB unreachable
P8	DB connection established
P9	Received OA OPC server data
P10	DB server data retrieved
P11	DB data synchronized

Table 3
OPC agent and OPC DB agent interaction – Petri net transitions

Transition	Description
T1	OA initialization
T2	OPC server available
T3	OPC server namespace available
T4	OA/DBA termination
T5	Initialize DBA
T6	OA/DBA termination
T7	DB unreachable
T8	DB reachable
T9	DB connection established
T10	OA/DBA termination
T11	Receive OA OPC server data
T12	Retrieve DB server data
T13	Data synchronization
T14	Retrieve DB server data
T15	OA/DBA termination
T16	OA/DBA termination
T17	OA/DBA termination
T18	OA/DBA termination
T19	OPC server not available
T20	OPC server namespace not available

O of each transition is presented in Table 7. Because this sample analysis is based only on a part of the Petri net, there are only $\#(\mu) = 16$ different markings that will be taken into consideration, that is $\mu = \{\mu_1, \mu_2, \ldots, \mu_{16}\}$. To simplify this analysis even more, only μ_{13} and μ_{14} markings will be selected.

Details of μ_{13} and μ_{14} markings can be observed in Table 4, Table 5 and Table 6. Each such marking μ_i is simultaneously an element of the set of worlds W. The relations sets R_i visible in Table 4 are elements of a greater relation set $R = \{R_1, R_2, \ldots, R_{16}\}$. For each world μ_i of W there is set of logical formulas Γ_i such that $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_{16}\}$. The last re-

Table 5
OPC agent and OPC DB agent interaction – μ_{13} and μ_{14} markings based on Fig. 2

Marking description	Marking (μ_i)
OPC srv available / OPC srv ns available / OA initialized / DBA initialized / DB reachable / DB conn established / Received OA OPC srv data	μ_{13}
OPC srv Available / OPC srv ns available / OA initialized / DBA initialized / DB reachable / DB conn established / DB srv data retrieved	μ_{14}

Table 4
OPC agent and OPC DB agent interaction – R_{13} and R_{14} relations sets based on Fig. 2

Markings (μ_i)	Relations ($r_{i:j}$)	Relation sets (R_i)
μ_{13}	$r_{13,1} = \{\mu_{13}, \mu_1\}$	R_{13}
	$r_{13,21} = \{\mu_{13}, \mu_{21}\}$	
μ_{14}	$r_{14,9} = \{\mu_{14}, \mu_9\}$	R_{14}
	$r_{14,18} = \{\mu_{14}, \mu_{18}\}$	

Table 6
OPC agent and OPC DB agent interaction – transitions for μ_{13} and μ_{14} markings based on Fig. 2

Marking (μ_i)	Logical formulas (Γ_i)	Transition (t_i)	Next marking (μ_j)
μ_{13}	Γ_{13}	t_2	μ_1
		t_5	μ_{21}
μ_{14}	Γ_{14}	t_4	μ_9
		t_{19}	μ_{18}

Table 7
Transitions input I and output O functions for the markings presented in Table 1

		Places										
		p_1	**p_2**	**p_3**	**p_4**	**p_5**	**p_6**	**p_7**	**p_8**	**p_9**	**p_{10}**	**p_{11}**
Input $I(t_1)$/ Output $O(t_1)$ functions	**$I(t_1)/O(t_1)$**	I	I	I	O							
	$I(t_2)/O(t_2)$	O			I	I			I			
	$I(t_3)/O(t_3)$		I / O		I							
	$I(t_4)/O(t_4)$			I / O	I							
	$I(t_5)/O(t_5)$				I / O	O						
	$I(t_6)/O(t_6)$	O			I	I			I			
	$I(t_7)/O(t_7)$					I / O	O	I				
	$I(t_8)/O(t_8)$					I	I	O	I			
	$I(t_9)/O(t_9)$					I /O	I / O		O			
	$I(t_{10})/O(t_{10})$	O			I	I			I	I	I	
	$I(t_{11})/O(t_{11})$					I / O	I	O	I			
	$I(t_{12})/O(t_{12})$						I / O		I / O	O		
	$I(t_{13})/O(t_{13})$						I / O		I / O		O	I
	$I(t_{14})/O(t_{14})$						I / O		I / O	I	I	O
	$I(t_{15})/O(t_{15})$						I / O		I / O		O	I
	$I(t_{16})/O(t_{16})$	O			I	I			I	I	I	
	$I(t_{17})/O(t_{17})$	O			I	I			I	I	I	
	$I(t_{18})/O(t_{18})$	O			I	I			I	I	I	
	$I(t_{19})/O(t_{19})$		I		I							
	$I(t_{20})/O(t_{20})$			I	I							

quired step relates to a valuation function ν that assigns objects from a domain of discourse to each term inside each logical formula of Γ_i such that $(\mu_i, \Gamma_i) \Rightarrow \mu_i \models \Gamma_i$. Logical formulas can be derived based on [30–32, 34] and thus are out of the scope of this work.

7. MAS infrastructure for DCS integration

Based on the presented ontology analysis, it became possible to implement a MAS-based DCS integration system. At the current development stage, the MAXS (Multi agent cross-platform system) platform consists of seven different types of agents organized hierarchically in three different logical layers, integrating a given DCS, as presented in Fig. 3. However, it remains open for further development as it is a flexible and scalable solution. MAXS agents include: the supervisory agent (SA), node agent (NA), management agent (MA), OPC agent (OA), discovery agent (DA), OPC database agent (ODBA) and management database agent (MDBA). The layered MAXS architecture is widely discussed in [35].

In short, SA is used to dynamically administer other platform agents that emerge in the platform during runtime, which means that it can terminate, suspend or create any platform agent. It can handle the process of agent relocation as well. DA is responsible for remote host environment discoveries. It acquires data about available OPC server services and creates an OA pool of agents. The number of OA agents created reflects the number of OPC servers discovered. Each OA is linked by DA to a single OPC server. OA is responsible for processing OPC server data. It configures an OPC server with OPC groups and OPC items. It reconnects locally or remotely between various different OPC servers, however it interacts with only one. As a result, OA propagates asynchronously gathered OPC server specific data to all available listeners. MA is responsible for user interaction with different OA agents. It is an agent with a GUI. MA provides OPC server specific data according to OA configuration and allows for process supervision and control through hierarchically structured, automatically synchronized OPC groups and an items tree. The number of MA agents varies over time because it is strictly related to the number of users operating the system. MA is designed to receive notifications from all available OA agents. ODBA and MDBA are responsible for configuring, establishing and maintaining a connection with the MAXS database on behalf of an OA or MA. They are used to store and retrieve OPC specific pieces of information. Both ODBA and MDBA can be used as redundant communication channels whenever the direct agent communication channel will become unavailable for an unknown reason.

NA is responsible for manual remote host registration in the existing MAXS platform. A complete description of the platform architecture and each platform agent has been presented in [36] and [37]. In addition to those agents, there are two further JADE specific agents. Those agents are DF (directory facilitator) and AMS (agent management system). The DF agent is an optional component that starts whenever JADE starts. AMS presence is required for any JADE platform to work properly. A complete overview of the DF and AMS agents can be found in [38].

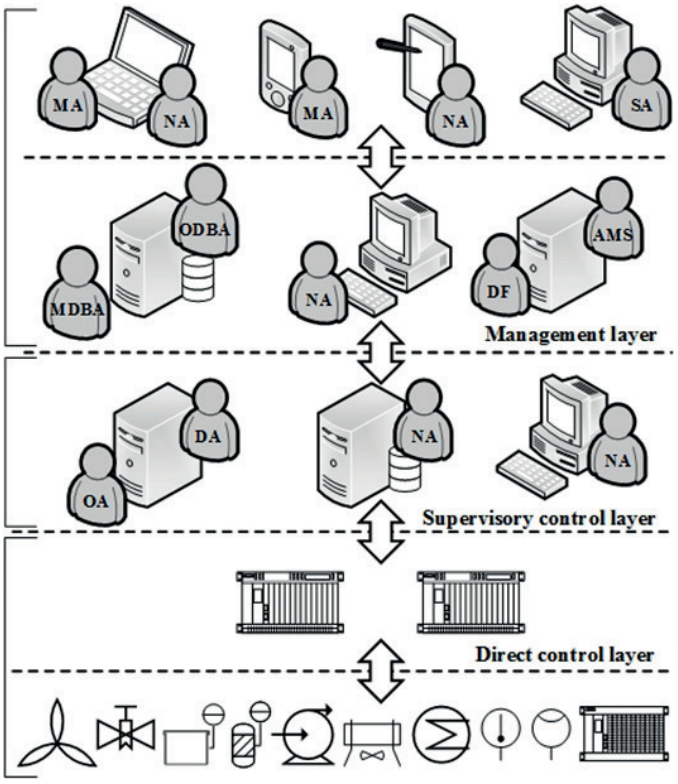

Fig. 3. MAXS – a solution to ontological MAS-based DCS integration

8. Conclusions

Ontological MAS (multi-agent system) is a software infrastructure capable of handling every problem related to the DCS (distributed control system) integration at the different stages of integration system manufacturing such as design, implementation and maintenance. Ontology-based MAS methodology has been known for quite some time now, and in the current situation it can already be supported by a wide variety of good quality, free or commercial development tools. The only problem that needs to be solved is the toolset content which will support the ontology-based MAS methodology during solution design, engineering and runtime. The toolset should be understood in terms of ready, "out of the box" software and analytical solutions.

In the presented approach, a small step has been made towards an explanation of such a toolset contents, presenting the meaning and usage of each particular element, and generating MAXS (multi agent cross-platform system) – a functional ontology-based MAS which is capable of integrating a dynamic DCS composed of various sources of data. In the presented approach to DCS integration, system engineering, UML (unified modeling language), XSD (extensible markup language schema definition) and source code are interchangeable and provide equal support during creation of an ontology representing domain knowledge and structure. Each such format can be reused simultaneously by an engineering team without the risk of synchronization problems because the established development

environment automatically supports maintenance of varied ontology formats. FOL is an analytic tool allowing to analyze not only ontology structure but ontology-based algorithms as well. To overcome the problem of FOL (first-order logic) undecidability, the FOL presented herein has been extended by means of modal logic. Such an approach to ontology analysis allows for studying it in terms of fixed set of worlds (states). This in turn allows for creation of a domain-based ontology maintaining the data integrity level in this direction, i.e. from the bottom up, and reusing well-known formats of various domain data structures.

Acknowledgements. This work was supported by the Ministry of Science and Higher Education under the BK-UiUA grant.

References

[1] V. Kulba, S. Nikolsky, and O. Zaikin, "Ontological approach to modelling of discrete event dynamic system", *Bull. Pol. Ac.: Tech.* 57 (3), 241–247 (2009).

[2] A. Byrski, R. Schaefer, M. Smołka, and C. Cotta, "Asymptotic guarantee of success for multi-agent memetic systems", *Bull. Pol. Ac.: Tech.* 61 (1), 257–278 (2013).

[3] D.W. Qian, S.W. Tong, and C.D. Li, "Observer-based leader-following formation control of uncertain multiple agents by integral sliding mode", *Bull. Pol. Ac.: Tech.* 65 (1), 35–44 (2017).

[4] G. Babin and W. Cheung, "A Metadatabase-supported shell for distributed processing and systems integration", *Knowl.-Based Syst,* vol. 21, no. 7, pp. 672–680, 2008.

[5] D. Monticolo, S. Mihaita, H. Darwich, and V. Hilaire, "An agent-based system to build project memories during engineering projects", *Knowl.-Based Syst.,* vol. 68, pp. 88–102, 2014.

[6] M. Moradi, A. Aghaie, and M. Hosseini, "Knowledge-collector agents: Applying intelligent agents in marketing decisions with knowledge management approach", *Knowl.-Based Syst.,* no. 52, pp. 181–193, 2013.

[7] J. Ashraf, E. Chang, O. K. Hussain, and F. K. Hussain, "Ontology usage analysis in the ontology lifecycle: A state-of-the-art review", *Knowl.-Based Syst.,* no. 80, pp. 34–47, 2015.

[8] G. Kalani, *Industrial Process Control: Advances and Applications*, Gulf Professional Publishing, 2002.

[9] X. Hong and W. Jianhua, "An extendable data engine based on OPC specification", *Computer Standards & Interfaces,* vol. 26(6), pp. 515–525, 2004.

[10] S. Cavalieri and F. Chiacchio, "Analysis of OPC UA performances", *Computer Standards & Interfaces,* vol. 36(1), pp. 165–177, 2013.

[11] X. Hong and W. Jianhua, "Using standard components in automation industry: A study on OPC Specification", *Computer Standards & Interfaces,* vol. 28(4), pp. 386–395, 2006.

[12] C. Sahin and E.D. Bolat, "Development of remote control and monitoring of web-based distributed OPC system", *Computer Standards & Interfaces,* vol. 31(5), pp. 984–993, 2009.

[13] OPC Foundation: Home Page, OPC Foundation, [Online]. Available: https://opcfoundation.org. [Accessed 13 August 2017].

[14] F. Iwanitz and J. Lange, *OPC – Fundamentals, Implementation and Application*, Huthig Verlag Heidelberg, 2006.

[15] J.M. Zamarreño, R. Mazaeda, J.A. Caminero, A.J. Rivero, and J.C. Arroyo, "A new plug-in for the creation of OPC servers based on EcosimPro© simulation software", *Simulation Modelling Practice and Theory,* vol. 40, pp. 86–94, 2014.

[16] V. Kapsalis, Ch. Fidas, and L. Hadellis, "Towards a Domain-Specific Context Acquisition, Presentation and Rule-Based Control Platform", *Int. J. Pervasive Computing and Communications,* vol. 9(1), 2013.

[17] J.L. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice-Hall, Inc., 1981.

[18] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*, Springer-Verlag Berlin Heidelberg, 2010.

[19] M.V. Ioradche and P.J. Antsaklis, *Supervisory Control of Concurrent Systems. A Petri Net Structural Approach*, Birkhauser Boston, 2006.

[20] E. Villani, P.E. Miyagi, and R. Valette, *Modelling and Analysis of Hybrid Supervisory Systems. A Petri Net Approach*, Springer-Verlag London Ltd., 2007.

[21] B.F. Chellas, *Modal Logic: An Introduction*, Cambridge University Press, 1980.

[22] M.J. Cresswell and G.E. Hughes, *A New Introduction to Modal Logic*, Routledge, 1996.

[23] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*, Wiley, 2011.

[24] F. Belardinelli and A. Lomuscio, "Quantified epistemic logics for reasoning about knowledge in multi-agent systems", *Artif. Intell.,* vol. 173, no. 9–10, pp. 982–1013, 2009.

[25] F. Belardinelli and A. Lomuscio, "First-Order Linear-time Epistemic Logic with Group Knowledge: An Axiomatisation of the Monodic Fragment", *Fundam. Inform.,* vol. 106, no. 2–4, pp. 175–190, 2011.

[26] W. Van der Hoek and M. Wooldridge, "Model Checking Knowledge and Time.," *SPIN,* pp. 95–111, 2002.

[27] F. Raimondi and A. Lomuscio, "Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams", *J. Applied Logic,* vol. 5, no. 2, pp. 235–251, 2007.

[28] A. Lomuscio and B. Woźna, "A Temporal Epistemic Logic with a Reset Operation", in *AAMAS*, 2007.

[29] B. Woźna, A. Lomuscio, and W. Penczek, "Bounded Model Checking for Knowledge and Real Time", in *AAMAS*, 2005.

[30] D. Berardi, D. Calvanese, and D.G. Giacomo, "Reasoning on UML class diagrams", in *Artif. Intell.* 168(1–2): pp. 70–118, 2005.

[31] L. Shan and H. Zhu, "A Formal Descriptive Semantics of UML", *ICFEM,* pp. 375–396, 2008.

[32] B. Beckert, U. Keller, and P.H. Schmitt, "Translating the Object Constraint Language into First-order Predicate Logic", in *In Proceedings, VERIFY, Workshop at Federated Logic Conferences (FLoC)*, 2002.

[33] B. Woźna and A. Lomuscio, "A Logic for Knowledge, Correctness, and Real Time", in *CLIMA*, 2004.

[34] D. Choiński, M. Senik, and B. Pietrzyk, "Ontology-based Management of a Network for Distributed Control System", *INFOCOMP,* pp. 97–102, 2014.

[35] D. Choiński and M. Senik, "Multilayer automated methods for the system integration", in *Luo, Y. (ed),* CDVE 2011, LNCS vol. 6874, pp 86–93. Springer-Verlag Berlin Heidelberg, 2011.

[36] D. Choiński and M. Senik, "Multi-Agent oriented integration in Distributed Control System", in J. O'Shea et al. (eds.), *KES-AMSTA* 2011, LNAI. Vol. 6682, Springer, Heidelberg, pp. 231–240, 2011.

[37] D. Choiński, M. Senik, "Ontology Based Knowledge Management and Learning in Multi-Agent System", in *KES-AMSTA* 2012, LNCS Vol 7327, Springer, Heidelberg, pp. 65–74, 2012.

[38] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & Sons Ltd, 2007.