Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science

Abstract and summary of doctoral dissertation

Ontology Adaptation for the Distributed Control Systems Management and Integration Purposes

Michał Senik

Promoter: drhab. inż. Dariusz Choiński

Gliwice, 2017
Contents

1. Thesis and Goal of Doctoral Dissertation ................................................................. 1
2. Introduction ............................................................................................................. 1
3. System Integration ................................................................................................. 2
4. Data Structure for the Distributed Control System .............................................. 3
5. Knowledge Representation for the Distributed Control System ....................... 4
6. Open Interface Protocol ....................................................................................... 5
7. Distributed Control System Integration ............................................................... 7
8. Ontology Based Interfaces and Data Structures .................................................... 8
9. Ontology for the Distributed Control System Integration ..................................... 9
10. Petri Nets for Modal Logic Ontological MAS Analysis ........................................ 11
11. Modal Logic Based Ontology Analysis – The Aim of the Work ......................... 16
12. Ontology Software Development Environment for the MAS DCS Integration .... 21
13. Software Development Infrastructure for the DCS Integration ............................ 22
14. Multi – Agent Infrastructure for the Ontology Utilization .................................... 24
15. MAS Infrastructure for the DCS Integration ........................................................ 26
16. Conclusions ........................................................................................................... 28

References .................................................................................................................. 31
Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>DCS – an idea of hierarchically layered system</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>OPC communication standardization</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Ontology formats transformations – general idea</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4</td>
<td>A part of Petri net marking graph based on the accessibility relation set</td>
<td>14</td>
</tr>
<tr>
<td>Figure 5</td>
<td>OPC Agent OPC DB Agent interaction</td>
<td>16</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Ontology based DCS integration – reused software modelling toolset</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7</td>
<td>MAS based DCS integration – reused software development toolset</td>
<td>24</td>
</tr>
<tr>
<td>Figure 8</td>
<td>OPC Server structure ontology modelling</td>
<td>25</td>
</tr>
<tr>
<td>Figure 9</td>
<td>MAXS layered structure – communication types</td>
<td>27</td>
</tr>
<tr>
<td>Figure 10</td>
<td>MAXS a solution to ontological MAS based DCS integration</td>
<td>28</td>
</tr>
</tbody>
</table>
1. Thesis and Goal of Doctoral Dissertation

Ontology is considered as an knowledge about particular domain. It is a static description of a domain structure and properties by means of which it is possible to define domain’s dynamic states, transitions and conditions of those transitions. Because of that it is possible to analyze such an ontology be means of a modal logic in predicate logic settings.

The goal of this doctoral dissertation is to create and analyze such ontology for the implementation of a Multi-Agent System (MAS) able to integrate different elements of a Distributed Control System (DCS).

2. Introduction

This work addresses the problem of an efficient approach to a Distributed Control System (DCS) integration engineering. This engineering is based on the knowledge gathered from the integrated system and modeled by means of an ontology. Engineering is an activity that can be broken down into several different areas of a science and technology. This requires a general knowledge, understanding and experience in multiple partially overlapping specialties. Because of that, very often, engineering of some software or hardware subsystem is distributed to a larger team. Modern engineering is strongly dependent on a different and often specialized software systems. Those different systems assist during designs creation, verification and predicting how things behave and work in the real environment conditions. Such an approach is of great value during creation of a design schematics and model generation of a different fundamental physical processes. By doing so engineering protects business by saving both time and valuable resources because, in most cases, there is no need to prepare time consuming and expensive prototypes of a physical processes. This allows to detect, analyze and correct on early stage of a product development possible faults along with its static and dynamic properties. Such an approach to engineering allows applying scientific discoveries by bringing as many innovations as possible available to the real world [1]. Successful engineering is an activity that tries to introduce the best workable solution for the specified requirements. It does so by evaluation and weighing different design choices on their merits, given that multiple possible approaches exists towards requirements realization. In such a case, engineering of a successful product requires identification and proper interpretation of each single requirements constraint. Proper engineering, prior to any full-scale production, tries to predict the quality of the established design. It is a responsibility that forces the engineering team to check whether the solution specification covers the given requirements and that the end result product will behave properly and will not cause any unintended harm. To assure that the end product complies with the design and meets requirements, engineering turns to prototyping, modelling and simulation and a wide
range of different tests such as destructive tests, nondestructive tests, and stress tests. Such an approach together with detailed analysis of an unsuccessful solutions helps during evaluation and detection of potentially risky, dangerous or erroneous elements of the obtained design.

3. System Integration

Presented approach to a Distributed Control System (DCS) integration engineering provides a solution to a problem of how to develop and maintain complex software integration systems over their lifetime in a dynamic and demanding conditions of the market in the various different branches of manufacturing industries. System integration is an activity that tries to bring together various smaller, incompatible hardware or software subsystems into a greater system. This is usually done in order to gather and combine different pieces of data together so that they can form a more comprehensive and meaningful structures for further sharing and processing. Such an approach usually requires creation of a new and specialized software system that will enable simple incorporation of those incompatible subsystems. However, it is often the case that the integration is harder to achieve the greater the number of an integrated subsystems are involved. This requires careful design and implementation of additional programming interfaces and services needed to assure simple flow of data between those different subsystems and a client endpoints. The difficulties arise mainly because of the considerable diversity of the specific solutions caused by the lack of coherent data representations used by a different vendors, technology and age of those products. That is why it is often very difficult to integrate each subsystem. Instead, it is a common thing to employ a short term strategy. This allows to build up gradually the integration system providing more functionality and responsibility in a more organized and consistent way by linking all the necessary subsystems when needed.

Distributed Control Systems are usually involved into some kind of production processes and without a proper approach to an integration the time between requirements formalization and the final product release can be unacceptably huge. Overcoming those problems require scalability and openness of the integration system. Establishing such characteristics allows performing adjustments to the existing integration system or introducing new functionalities in a reasonable timeframe. They allows maintaining the cost and the quality of the released products, protecting interests of each end client and the enterprise itself. Moreover, they protects the enterprise from experiencing the problem of costly and time-consuming re-investments by reusing the full potential of the existing production infrastructure up to its limits. However, this flexibility is not achieved easily and the solution to this problem is not an easy task. One major reason such problems do exist are the technological boundaries, inabilities and tendencies limiting engineering capabilities and runtime functionalities of the resulting DCS integration system. In the proposed approach, an attempt towards establishing a reasonable solution
to the described problems was made by introducing an ontology based DCS integration engineering from both methodological and technological perspectives.

Hardware and software infrastructures coexists together as the data originating in the hardware has to be somehow delivered, processed, interpreted, stored and presented in a meaningful and understandable way. This task is usually performed in the software infrastructure. Because of that, those both infrastructures has to be treated as a one integration system. Each such infrastructure can evolve independently over time. In terms of the Industrial Automation (IA), a Distributed Control Systems (DCS) are an example of such systems. A well-designed DCS is a future proof solution that characterizes with interconnectivity and interoperability that directly impacts system openness, scalability and flexibility. However, those factors are hard to obtain, thus, there are many DCSs that are technologically limited or bounded to multiple new or legacy subsystems[2]. DCSs evolution is a normal thing, especially with regards to the demanding and rapidly changing market conditions. Each company, has to meet some deadlines, achieve planned target sales and stay competitive at the same time. That is why, for each such company, it is very important to invest continuously to their production system a considerable amount of their valuable resources. Those investments has to be maintained at reasonable and satisfactory level such that the quality of manufactured products would not decrease[3].

4. Data Structure for the Distributed Control System

DCSs are usually heterogeneous, which means that they involve a great number of a different subsystems. Heterogeneity is a compositional relationship amongst numerous interconnected and interoperable third party subsystems that are usually distributed geographically and are a source of a various pieces of information. Heterogeneity of a DCS integration system is possible through a standard conforming hardware or software elements. This means that each such element use a well-known, common interface that allows for an 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. DCS integration systems can additionally present autonomy, intuitiveness and loosely coupled structure. Autonomy of DCS integration systems decides about its quality and can be especially helpful in a variety situations including self-maintenance, data gathering, analysis and processing, fault detection, system diagnosis, alarms and events notifications, reporting, planning and configuration. This is true, because integration systems, during runtime, have to efficiently assist human operator in processing of a large quantities of a different data in a relatively short time. This process is usually referred to as knowledge management (KM). Intuitiveness decides about the quality of how an integration system cooperates with the human user. It directly decides about the learning curve that tells how fast a maximum potential of the integration
system can be drawn by an engineering team. Loosely coupled structure relates to the notion of how well software components, forming integration system are separated one from each other. It is a degree of direct knowledge a software components exposes. A well-designed software component, in terms of a software design, has to maintain a considerable level of independency, providing only process related data through a well-known communication interface, leaving all its implementation details hidden, out of the scope of other components. This is a one of many key concepts of the design patterns [4][5]. Such an approach enforces interface based implementation, which is a very important concept whenever a software component has to be modified or replaced. Maintainability directly derives from the loose coupling notion. It decides about the degree of modification that can be performed inside each software component, forming integration system, without having the problem to modify or even replace still working parts of the integration system. This maximize integration system efficiency, reliability and safety extending its lifespan with regards to the changing environment[2].

5. Knowledge Representation for the Distributed Control System

Knowledge management relates to the notion of capturing, developing, sharing and using a domain knowledge. This knowledge is enclosed inside each DCS and is used to assists human operator in the decision making processes. However, the quality of how this process is performed by a different DCSs varies, thus each company is particularly interested in an autonomous, intelligent, intuitive and maintainable integration systems characterizing with interoperability and interconnectivity that can manage the domain knowledge best. This is true, because such a system has to respond to users’ requests or environmental changes and take initiatives, attending users’ needs [3][6]. Domain knowledge similarly to the integration system itself is heterogeneous and distributed as it relates to a different system parts, thus, for this knowledge to be meaningful, unambiguous and useful, there is a strong need to use a common terminology. This is achieved by means of an ontologies that can be used during integration system design, runtime and maintenance [3][7]. Terminology is a discipline that focuses on terms and their use. Terminology studies the development of terms and their interrelationships within a specialized domain. Terms are simple, compound or multi word expressions. Those expressions has specific meanings in specific contexts and can deviate from the meaning established in common, everyday language. DCSs present strong, hierarchical, layered design [8]Figure 1. Interaction between each layer, depending on a careful selection of communication interfaces and data structures, is a key factor deciding about DCS’s overall quality and efficiency [9]. Because of that, various different Industrial Automation (IA) vendors of control and instrumentation systems are gradually investing more valuable resources in various technological researches, 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 focus on and contribute to standardized solutions rather than design proprietary, costly, and private ones. Having standardized solutions assures simplification and reusability of control system interfacing, at the same time increasing the enterprise’s long term competitiveness, recognizability and income [8][10][11]. Open Platform Communication (OPC) is a great example of such an approach and there are a lots of studies concerning practical utilization of OPC such as [11] and [12] to name a few.

Figure 1  DCS – an idea of hierarchically layered system

6. Open Interface Protocol

OPC is an open automation interface protocol actively developed by the OPC Foundation since 1994 [13][14]. Initially, the OPC Foundation comprised of only few automation vendors such as Fisher-Rosemount, Intellution, Opto 22 and Rockwell Software. However, now, according to the OPC Foundation’s website, it comprises of many additional members from all over the world [14][15]. OPC was established as a method for a fast and efficient data retrieval, integration, analysis and communication between various different, third party automation devices and heterogeneous subsystems [15][16]. From a technological point of view, it is a standard responsible for establishing the convenient and efficient communication between various automation components and control hardware and field devices [14]. It is designed
to provide the most suitable conditions for an easy integration between different office products and information systems on the company level including Enterprise Resource Planning (ERP) and Manufacturing Execution Systems (MES). Resulting status and process data can be presented, edited and saved using different formats for further processing in a production planning system. Manufacturers of different OPC components are not limited to their solutions (complexity), as long as all appropriate OPC specific definitions of the interfaces, and methods and rules regarding their implementation, are maintained. As a result, the most diverse OPC components developed by different manufacturers after configuration can freely work together without the need of additional programming effort with regards to the communication interfaces. Moreover, such an approach allows solving the problem of having complex dependencies existing between software and hardware components. This is true because OPC hides each smallest implementation detail behind its abstract interfaces, supporting that way notions of interconnectivity and interoperability such that complete OPC compliant hardware and software components can be easily exchanged. Moreover, due to the standard communication interface existing between the client and the server hosting various process data, the communication is completely independent of the physical data source [15]Figure 2.

Until now the OPC Foundation introduced a set of various standard interfaces designed for a different fields of application in automation technology. Each such interface enables an easy and efficient data exchange between OPC compliant hardware and software components developed by different automation vendors. Because of the fact that OPC specifications can be applied in different fields of automation applications they are largely independent from one another, however, it is straightforward task to combine them together in one 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].

Figure 2 OPC communication standardization
7. Distributed Control System Integration

In the world of IA DCS integration, the level of complexity and expectations towards the flexibility of the obtained solution is constantly rising, which is why terms such as interconnectivity and interoperability between various different integrated subsystems are very important [15]. Bringing together, configuration and customization of a wide array of various, new or existing, usually incompatible hardware or software, 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 a DCS integration. It is a complex, time consuming, most challenging and demanding task, which involves significant amount of diagnostics, troubleshooting and software engineering work, requiring a lot of skills, talent, broad knowledge and real life experience in a variety of integrated hardware and software subsystems. The nature and the amount of needed work is overwhelming, as it involves all the activities ranging from the design and implementation, to documentation, deployment and maintenance. It is worth mentioning that compatibility of both hardware and software components is always a key element that directly impacts the speed, quality, sustainability, openness, robustness, scalability and the maintainability of a resulting system [10]. That is why DCS integration is a very complex task, taken into account that substantial number of currently existing different subsystems were designed with rather low awareness about future connectivity possibilities. This is a rather unfortunate approach since it not only lowers future DCS Integration System (IS) quality but also contradicts the notions of interconnection and interoperability of subsystems, unarguably disqualifying the obtained system as being an open, robust, scalable and maintainable solution. One of the major problems that needs to be answered during DCS integration is how to format, pass and process the obtained data. This is true, because DCS integration engineering focuses around the notion of harnessing into one complete picture all available information originating at various different points. However, the acquisition of those different pieces of information cannot be achieved without a suitable communication interface.
8. Ontology Based Interfaces and Data Structures

DCS integration focuses on a designing or reusing already existing communication interfaces. Establishing the right data structure and a corresponding communication interface is a crucial matter, because in this way all gathered data can be presented in a meaningful, consistent and easy to process format which can be understood by both man and machine [9]. Moreover, it can improve the obtained system cost, quality, sustainability, openness, robustness, scalability and maintainability. However, in order to assure interconnection and interoperability of the obtained system in the long term, standardization of both data structure and corresponding communication interface has to be taken into consideration [9]. To be effective, standardization has to be strictly maintained and followed throughout the entire process of DCS integration system engineering, starting from its lowest towards the highest levels. In reality, however, compatibility between data structures and communication interfaces is followed more or less strictly only on the lowest levels of the integrated DCS, which is where various pieces of raw information originate. This is an inappropriate approach, since it leaves a lot of freedom during the rest of the DCS integration giving a lot of opportunities for subjective interpretations and judgments leading to very complex, inefficient and error prone solutions. Such situations happen very often, causing a lot of confusion inside engineering teams, unnecessarily stretching already overloaded project budget up to its limits and potentially putting at risk the success of the entire enterprise. However, through a proper communication interface reusability, shareability, interconnection and interoperability of different hardware or software components is possible. Only a binding mechanism between data structures and communication interfaces, such as ontology, supporting an idea of standardization, can truly assure interconnectivity and interoperability between distinct subsystems of an integrated DCS.

In general, an ontology is considered as a system knowledge. This knowledge is modeled and implemented beforehand DCS integration system development and it is known prior to its runtime [17][18]. Ontology describes static properties of the DCS integration system as well as runtime states in which the system can reside and conditions of transitions between those states. Based on the ontology, different parts of the DCS integration system can share their knowledge and work together over the system problems. This is true because knowledge sharing is the natural way for DCS integration system components to solve assigned problems and achieve their goals. Ontology, amongst other things, is composed of a set of simple and more complex rules such as concepts, predicates and actions entities. Those entities can be organized in a hierarchical structures which means that simpler ones can be nested in more complex ones. Such an approach allows for a dynamic management of the DCS integration system knowledge using its static descriptions.

Ontology is a good medium through which integrated DCS domain knowledge can be easily divided, ordered, categorized and shared. Ontologies help to disambiguate and unify domain
knowledge by precisely defining the meaning of each individual symbol, the relations existing between them, the assignment of each symbol to each domain object and precisely localizing each individual object in the hierarchical domain structure. The main difference between ordinary data models and ontologies is their level of generality, reusability, portability and interoperability. Data models are usually task specific and implementation oriented while ontologies are more generic and task independent [19]. Moreover, it is important to mention that an ontology by itself is only a static element, and for it to be reused, during runtime, a suitable infrastructure has to be utilized, one which supports the notions of both interconnectivity and interoperability, such as for example Foundation for Intelligent Physical Agents (FIPA) [20][21] based Multi-Agent System (MAS).

9. Ontology for the Distributed Control System Integration

Ontologies are always about some specific domain terms, their hierarchical or compositional relations, their meaning, logic rules and vocabulary, and they specify those features explicitly, in an elaborate way, using specific representation languages. Ontologies are 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. Because of that, an ontological vocabulary differs from a human oriented one. It is more expressive and cognitive since it explicitly defines additional logical statements, allowing 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 and need of 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 applying composition and inheritance mechanisms efficiently. Class notion is both human and machine interpretable, and supports easy, bidirectional conversion to 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. A software class, similarly to a concept but in a formal and explicit way, represents each tiniest, domain characteristic
element along with its hierarchy and composition. This means that much like a software class, a concept can be either atomic or composed of many different concepts. Additionally, it can take part in many different, greater hierarchies of concepts. A class hierarchy is a classification of various, domain specific object types, and refers to the notion of specialization of classes in which one class is a parent, base class from which other classes (called subclasses) derive base characteristics. Class composition refers to the notion of combining simpler objects’ types into more complex structures, and is a key building block of various, domain specific types of object’s structures. 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.

In the presented solution, Eclipse IDE along with various mutually compatible plug-ins assists during ontological MAS DCS integration system development. Eclipse is a very popular development platform that can be reused during software engineering activities, such as UML modelling, implementation and debugging. Eclipse, combined with different compatible plug-ins, is reused during ontology development, allowing for an automatic, bidirectional conversion between UML class diagrams, XSD, XML and source code. Such an approach allows for better ontology development because different engineers with different fields of expertise can choose among different ontological formats to share their knowledge most efficiently and synchronize with the existing ontology. Given the above, it is worth mentioning that XML is used mainly during runtime as a communication medium between various MAS DCS integration system agents. XML is only an instantiation of its parent XSD. The XSD serves as a validation medium and a blueprint for each obtained XML message. Additionally, the ontology XSD format
is used as a platform independent medium through which it is possible to recreate both ontology UML class diagram and its source code representation as well.

In the presented approach to ontological MAS DCS integration system engineering FOL is used as an additional tool for the ontology analysis. FOL plays important role especially during modelling, validation and offline analysis of various different ontological algorithms. Using FOL formulas, supporting ontology based UML model, requires introduction of an OO domain vocabulary, which contain a few constants, functions, unary binary, ternary and n-ary predicate symbols. However, proper FOL analysis requires some clarification of how to properly interpret especially binary and n-ary FOL predicates. Each predicate is assigned a self-interpretable name and contains an ordered number of arguments therefore it is important to analyze each such predicate from the leftmost to the rightmost argument. The name of the predicate is a symbolic abbreviation of the logical statement about the arguments list. Complete description of each such predicate is as much important as each derived FOL truth statement. Thus, without such description no valid FOL analysis is possible.

10. Petri Nets for Modal Logic Ontological MAS Analysis

A Petri net is a discrete event dynamic system and a mathematical modeling language allowing the description of a complex, concurrent and distributed systems[22][23][24][25]. Its structure C can be described as a tuple of four elements:

\[ C = (P, T, I, O) \]

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. The input function I maps each transition \( t_j \) directly with a corresponding collection of input places \( I(t_j) \). Similarly to the input function, the output function O maps each transition \( t_j \) directly with a corresponding collection of output places \( O(t_j) \). Both places P and transitions T sets are mutually disjoint. An element of a set of places P is denoted by a \( p_i \), \( i = 1, 2, ..., n \), and an element that belongs to a set of transitions T is denoted by a \( t_j \), \( j = 1, 2, ..., m \). Single place \( p_i \) can be considered as an input place of a \( t_j \) transition if it directly belongs to a collection of \( t_j \) transition input places I. It can as well be considered as an output place of a \( t_j \) transition if it directly belongs to a collection of \( t_j \) transition output places O. It is worth mentioning that both collections can have multiple occurrences of the same element such that it is possible to obtain a multiplicity of an input place \( p_i \) for a transition \( t_j \) and a multiplicity of an output place \( p_i \) for a transition \( t_j \).

Petri net structure C can be supplemented with a graphical representation \( G = (V, A) \) consisting of circles and bars. Each single circle represents a place \( p_i \). Each single bar represents a transition \( t_j \). Together they form a set of vertices V. The vertices set V is formed
by a two disjoint sets of places $P$ and transitions $T$. Circles and bars are connected with directed arcs $A$. An element of a vertices set $A$ is denoted by an $a_i$, $i = 1, 2, \ldots, r$, in which case an directed arc is considered as a tuple of two elements that belongs to the vertices set. Some arcs are directed from a place $p_i$ to a transition $t_j$ and some are directed from transition $t_j$ to a place $p_i$ which means that either $v_j \in P$, $v_k \in T$ or $v_j \in T$, $v_k \in P$. This refers to the notion of input and output places of a transition $t_j$. Moreover, because Petri net is a directed multigraph each transition $t_j$ and place $p_i$ can have multiple arcs going in and out.

Another important feature of a Petri net is its marking $\mu$ which is an assignment of so called tokens to the places $p_i$ of a Petri net. A tokens are usually represented by a small dots on a graph. A marked Petri net $M$ is referred to as structure:

$$M = (C, \mu).$$

Petri net marking is a function that relates places $P$ with nonnegative integers $N$. Tokens reflects the dynamic nature of a modeled system. Tokens 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 a positions can change during the execution of Petri net. In a more detail the marking can be defined as vector consisting of $n$ elements. In such a case a marking vector $\mu$ gives a total number of tokens for each place $p_i$. It is worth mentioning that the number of tokens that a single place $p_i$ can hold is unbounded. Execution of a Petri net depends on so called transition firing during which transition $t_j$ removes tokens from its input places and creates new ones in its output places. A transition can fire if it is enabled and is enabled if it has at least as many tokens in each of its input places as the number of arcs from the input places to the transition. Each time an enabled transition $t_j$ fires a change in a Petri net marking $\mu$ occurs resulting in a new marking $\mu'$. Petri net execution stops when there are no enabled transitions available.

Petri net marking defines its state. Each time an enabled transition fires a change in a Petri net thus a change in its state occurs. This change can be described by a two argument $\delta$ next state function. The result of the next state function is a new marking $\mu'$. 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 $\delta$ next state function. For the purposes of the further modal logic analysis it is important to note that each Petri net marking $\mu_i$is considered as a separate world inside which given logical formulas will be examined.

Given above, a marked Petri net can be further characterized using notions such as reachability, liveness and boundness. Reachability problem tries to identify whether a given Petri net marking can be achieved. Liveness tries to answer the question about the conditions under which each Petri net transition can be fired. Based on that a Petri net can consist of dead, potentially fireable and infinitely often fireable transitions. Boundness provides an information about the number of tokens that can be observed in each Petri net marking. Together, the results
obtained from the reachability, liveness and boundness analysis can yield many interesting
and important pieces of information about engineered system i.e.: whether there are some states
that cannot be reached, are there any transitions that cannot fire, are there any deadlocks,
is the Petri net reversible or is there a chance for a memory leakage.

Classical modal logic extends classical propositional and predicate logic and includes
additional possibility ◇ and necessity □ operators that expresses modality. Both operators
are unary and can be expressed in terms of the other with negation:

◇P ↔ □¬P
□P ↔ ◇¬P.

The most often used modal logic system is S5 although there are additional systems.
The S5 system defines the necessity and possibility modalities. Based on S5 a proposition
is necessary if it is true at all possible worlds. A proposition is possible if it is true only at some
possible worlds. Given that it can be stated that there can be many things that are true at various
possible worlds. However, if something is true at each world it is said that it is necessary.
On a contrary, if something is possible at 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 a two
non-empty sets that will contain possible worlds W Table 1 and relations R Table 2 that exists
between the members of a possible worlds set.

∀ µ,∃ W. µ ∈ W.

∀ R,∃ R. R ∈ R.

∀ r_{ij} ∈ R, r_{ij} ∈ R \land r_{ij} = \{ µ_{i}, µ_{j} \} \land µ_{i} R µ_{j},

∀ µ_{i} R µ_{j}∃ t_{ij} ∈ T ∧ µ_{i} R µ_{j} ⇒ δ(µ_{i}, t_{ij}) = µ_{j}.

Based on the above it can be said that the state of a world µ_{i} is a direct possibility for world
µ_{i}, i.e.: µ_{i} R µ_{j} Figure 5. The last step in the initial analysis stage is determination of a set
of a logical formulas Γ true for each world µ_{i}, Figure 4.

∀ Γ,∃ Γ. Γ ∈ Γ.

∀ µ,∃ Γ_{i}(µ), Γ_{i} ∈ Γ_{i}.

∀ µ,∃ Γ_{i}(µ) ⇒ µ_{i} = Γ_{i}.
The \( \nu \) symbol is called a valuation function that assigns objects from a domain of discourse to each term inside each logical formula of \( \Gamma_i \). Together, non-empty set of worlds \( W \), their relations \( R \) and valuation function \( \nu \) form a model \( M \) of an engineered system:

\[
M = (W, R, \nu).
\]

Figure 4 A part of Petri net marking graph based on the accessibility relation set

<table>
<thead>
<tr>
<th>Place</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>OA Initial Inactive State</td>
</tr>
<tr>
<td>P2</td>
<td>OPC Server Available</td>
</tr>
<tr>
<td>P3</td>
<td>OPC Server namespace available</td>
</tr>
<tr>
<td>P4</td>
<td>OA Initialized</td>
</tr>
<tr>
<td>P5</td>
<td>DBA Initialized</td>
</tr>
<tr>
<td>P6</td>
<td>DB reachable</td>
</tr>
<tr>
<td>P7</td>
<td>DB unreachable</td>
</tr>
<tr>
<td>P8</td>
<td>DB Connection established</td>
</tr>
<tr>
<td>P9</td>
<td>Received OA OPC Server data</td>
</tr>
<tr>
<td>P10</td>
<td>DB Server data retrieved</td>
</tr>
<tr>
<td>P11</td>
<td>DB data synchronized</td>
</tr>
</tbody>
</table>

Table 1 OPC Agent OPC DB Agent interaction – Petri net places
<table>
<thead>
<tr>
<th>Transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>OA Initialization</td>
</tr>
<tr>
<td>T2</td>
<td>OPC Server Available</td>
</tr>
<tr>
<td>T3</td>
<td>OPC Server namespace available</td>
</tr>
<tr>
<td>T4</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T5</td>
<td>Initialize DBA</td>
</tr>
<tr>
<td>T6</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T7</td>
<td>DB unreachable</td>
</tr>
<tr>
<td>T8</td>
<td>DB reachable</td>
</tr>
<tr>
<td>T9</td>
<td>DB Connection established</td>
</tr>
<tr>
<td>T10</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T11</td>
<td>Receive OA OPC Server data</td>
</tr>
<tr>
<td>T12</td>
<td>Retrieve DB Server data</td>
</tr>
<tr>
<td>T13</td>
<td>Data synchronization</td>
</tr>
<tr>
<td>T14</td>
<td>Retrieve DB Server data</td>
</tr>
<tr>
<td>T15</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T16</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T17</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T18</td>
<td>OA/DBA termination</td>
</tr>
<tr>
<td>T19</td>
<td>OPC Server not available</td>
</tr>
<tr>
<td>T20</td>
<td>OPC Server namespace not available</td>
</tr>
</tbody>
</table>

Table 2  OPC Agent OPC DB Agent interaction – Petri net transitions
11. Modal Logic Based Ontology Analysis – The Aim of the Work

Propositional logic satisfiability is decidable but not fully capable to address all the characteristic properties of the complex systems such as a MASs. This is true even though propositional logic allows to build knowledge-based agents capable of answering queries about the domain by inferring new facts from the existing, known ones. The domain that is to be represented and reasoned about consists of a number of objects with variety of properties and relations among them. Propositional logic represents only the 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 or even impossible to encode.
This is because the statements about the domain objects and relations existing among them needs to be enumerated. The problem is that such knowledge base (KB) grows uncontrollably large. Possible solution is to introduce variables and allow 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 an arbitrary objects. It introduces also quantifiers which allow making statements about groups of objects possible without the need to represent each of them separately. However, the satisfiability and the decidability of the FOL formulas, thus their validity cannot be clearly verified. In mathematical logic any formula is a fixed sequence of logical and non-logical symbols belonging to a specific alphabet of a formal language. Formula satisfiability is achieved if it is possible to find such an interpretation under which the formula is true. Logical formula is valid if the truth of the conclusion is guaranteed by the truth of the premises. In other words, for a valid formula it is required that if each premise evaluates to true a conclusion evaluates to true as well. Logically valid formulas are often referred to as theorems. In general, an interpretation can be considered as an assignment of meaning to the formal language alphabet non-logical symbols. By doing so it is possible to evaluate the truth of each logical formula in a formal language. If under given interpretation a logical formula evaluates to true, the interpretation is considered as a model of that formula. It is worth mentioning that each FOL language is defined explicitly by a signature which consist of an infinite set of variables and different logical, and non-logical constant, function or predicate symbols. Interpretation assigns a meaning from a non-empty domain of discourse D to each non-logical symbol existing in the formal language. Given that, interpretation assigns an element of D for every constant, n-ary function from D to D for each n-ary function and n-ary relation on D for each n-ary predicate symbol. Such an interpretation object is usually referred to as a structure. In mathematical logic decidability refers to the problem of existence of so called effective method or an algorithm that allows to evaluate a truth of a logical formulas describing the system in a fixed amount of steps or time. It is a mechanical method which is dedicated to solve problems existing only in a specific class. Effectiveness of this method has to be measured with respect to the parent class of problems because a specific method can be effective in one class and less effective or not effective at all in another. In general, an effective method, when applied to its class of problems characterize with a finite number of steps after which it terminates and produces an answer. It is worth mentioning that an effective method does not necessarily be automatic. It can as well be semi-automatic or completely manual and be performed without additional assistance of other automatic or semi-automatic software analytic tools.

In the propositional logic an interpretation of a formula is simply an assignment of truth values to each atomic sub-formulas, however, in the FOL this is not a case. In the FOL, to define an interpretation of a formula, it is expected to specify a domain of interest along with an assignment to constant, function and predicate symbols. Additionally, for each such
interpretation, evaluation of a given formula should also be possible. If a formula is true under a given interpretation, this interpretation is said to be a model for a formula. In propositional logic some formulae are tautologies because they have the property of being true under all given interpretations. Moreover, there is a procedure based on the truth tables which can be used to tell explicitly whether any propositional formula is a tautology. Additionally, in most cases, deriving a complete set of interpretations for a propositional formula is a quite straightforward task. This is true because each such formula is composed of a smaller, atomic sub formulas that can be only either true or false. However, in FOL this is not the case. In FOL it is impossible to use truth tables to determine whether a formula is valid. In general it is also impossible to enumerate a complete set of possible interpretations. This is because a FOL formula consists of various constant symbols and variables. For that reason FOL is a source of undecidability. This is true, because there does not exist such a procedure that will guarantee, in a finite amount of time, whether a FOL formula is valid or not, thus a different approach is required.

From an analytical perspective one such solution is the modal logic [26][27][28]. Modal logic is an extension of a classical propositional logic. However, it can also be studied in the FOL settings [29][30]. In modal logic, evaluation of formulae occurs within a fixed set of worlds, rather than a single world. Thus, it is possible that a formula may be true in some worlds and false in the 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. In modal logic setting, satisfiability and decidability of a FOL formulas can be verified on per world basis, without having the problem of enumerating each interpretation [29] because it is possible to find such world and such interpretation under which this formula is true [31]. Such interpretation is then referred to as a formula model [32]. Given that, a set of logical formulas, under particular interpretation can be used as a logical specification that describe a dynamic system in terms of a single worlds or more precisely states. Such approach is correct given that evaluation of a FOL formulas is based on the modal logic and focuses on the certain set of worlds. This reflects a finite state characteristics of a complex systems such as a MASs.

Terms such as validity, satisfiability and decidability are particularly important whenever it comes to the model checking. In the modal logic terms, a model of a logical formula is referred to as an ordered triple:

\[ M = (W, R, \nu) \]

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 \( \nu \) that, under given interpretation, binds domain specific objects with each term inside each logical formula ensuring formula satisfiability in each possible world. The accessibility relation expresses that the state of affairs of a one world is reachable or possible for another world. Such an approach allow 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 (states) and relations (transitions) that exists in 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 technique. In this approach verified system $S$ can be represented by a logical modal [32] model $M$ and a verified statement $P$ is expressed by a logical formula $\Gamma_P$. Consequently, a verification via a model checking [32][33]is an activity that tries to provide an answer whether or not:

$$M \models \Gamma_P.$$

In the modal logic model checking the decision problem considers whether a logical formula, which represents a part of specification of a particular system is satisfiable in the model that represents each possible evolution of this system [29][34].

Model checking is a very important notion because it allows establishing a design of a system compliant with certain, desirable properties obtained directly from the system specification. This specification characterize the system functionality in terms of forbidden and allowed states. In such a case, if a system does not fulfill one of its specification properties, a defect is found. The system is correct if it satisfies each property given by the specification. However, it is worth mentioning that this correctness is not absolute as it is always relative to the system specification [35]. Thus, even a correct system can contain defect considering the possibility of frowned specification. The details of the model checking problem has been widely described in the scientific literature in terms of propositional (classic), temporal modal logic [31][32], epistemic (knowledge based) [29][30]and quantified (first order) logic[36].

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 in a modal logic settings. During initial design stage, ontology is created by means of an 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 delivered solution. To avoid such situation a verification is required. In many latest researches First Order Logic (FOL) is recognized as a suitable analytical tool that has a sufficient power to perform a successful UML class diagram verification [37][38][39]. However, in general, FOL satisfiability is not decidable thus UML based FOL model cannot be clearly verified. A possible solution to this problem lays in the modal logic. In this 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 (specification). Consequently, original problem of ontology verification becomes a problem of a formal verification of a model of an UML based FOL formulas in the modal logic settings. Ontology when ready will be used during implementation and eventually during runtime of a MAS based DCS integration system.

Each MAS based system is composed of a different types of agents that during runtime perform different operations on a behalf of other agents. Agents by achieving their smaller goals achieves together greater goals of their parent system. Each agent during the design can be described by means of a Petri Net with which it is possible to differentiate each single state in which agent can reside, transitions between those states and conditions of those transitions. By doing so it is possible to obtain indirectly 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 an UML based FOL formulas in the 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[34][40]. Given that it is possible to put into motion the idea of formal ontology based UML model verification using model checking in the 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 a formal ontology based UML model verification allows to capture not only the structural inconsistencies in the UML class diagram properties, composition or hierarchy but functional inconsistencies of each agent type that forms 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 on a part of an ontology. Therefore the entire process of a 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.
12. Ontology Software Development Environment for the MAS DCS Integration

In the presented approach to ontological MAS DCS integration engineering, development is based on both Microsoft Visual Studio [41] and Eclipse [42] IDEs, however, most of the engineering work is performed in Eclipse IDE. This is mostly due to the fact that Eclipse IDE can be easily integrated with various out of the box, free and mutually compatible tools. In the presented approach, plugins such as UML Designer [43], UML2JavaGenerator [44], Jar2Uml [45], Java Architecture for XML Binding (JAXB) [46], Hypermodel[47], Eclipse Modeling Framework (EMF) Generator Model[48] and EclipseLinkMOXy (MOXy) [49] are required. The suggested toolset presents a great level of compatibility, allowing for an easy and automatic, bidirectional conversion between different ontology formats, without the problem of loss of data and integrity.Figure 6.

The UML Designer plugin is a graphical tool, which allows both editing and visualization of: UML package hierarchy, class, component, composite structure, deployment, use case, activity, state machine, sequence and profile diagrams types. UML Designer provides a nice and intuitive User Interface (UI) allowing for easy and efficient software development. In the presented solution, UML Designer is used mainly to model both class and sequence diagrams. The UML2JavaGenerator plugin is compatible with the Eclipse UML2 project and fully integrates with UML Designer, allowing for automatic Java code generation. The Jar2Uml plugin allows for an automatic conversion of either Java jar files or Java class files into their corresponding UML model.

Figure 6 Ontology based DCS integration – reused software modelling toolset
The JAXB plugin allows for an automatic and bidirectional conversion of XSD to Java classes. EclipseLinkMOXy enables efficient binding between Java classes and XSDs. MOXy supports JAXB by allowing it to provide mapping information through annotations, and support for storing the mappings in XML format. XML mappings enable handling the complex XML structures without having to mirror the schema in the Java class model. The Hypermodel plugin allows for an automatic conversion directly from an XSD to an UML class diagram, creating as a result a fully functional and compatible model that can be easily viewed and edited in UML Designer. The EMF by itself is a modeling framework and code generation facility designed for various building tools and other applications. It is based on a structured data model without which mutual compatibility of the mentioned toolset could not be achieved. In the presented solution, the EMF Generator Model is used. By default, it is one of the features delivered along with the EMF framework allowing for an automatic code, XSD and UML generation based on either annotated Java code, UML or XSD. The resulting XSD ontology becomes a format that can be easily used to perform automatic migration between Java and .Net platforms. Such an approach allows maintaining mutually compatible source code bases between two different programming platforms. To perform such automatic translation, a Microsoft Visual Studio, proprietary .Net Xsd.exe [50] or Xsd2Code [51] can be used. Both present similar capabilities. However, the main difference between them is that Xsd.exe comes, by default with the .Net platform and can be invoked only from command line. The Xsd2Code is an additional tool that after installation integrates with the Microsoft Visual Studio IDE providing nice and easy GUI to configure output.

The established toolset allows also for efficient and concurrent ontology development. Different engineers, with different technical experience, skills and insight can share their knowledge about future ontological MAS DCS integration system using the ontology as their collaboration medium. That is why it is important to have a unified and a fully compatible ontology development environment.

13. Software Development Infrastructure for the DCS Integration

Developed by the authors over a couple of years MAXS (Multi Agent Cross-Platform System) is a real time processing, hierarchical, multilayered, MAS capable of dynamic adjustment to the existing integrated system conditions[52]. The main advantage of the hierarchical, multilayered. Structure of the presented MAS is interoperability between its layers, assuring technology-independence [9][10]. From the programming perspective, MAXS base on various different software development tools allowing it to reuse different OS platforms during runtime, and extending its integration capabilitiesFigure 7. MAXS wraps over JADE [21][53] using a set of newly implemented agents, and creating a complex, distributed agent system. MAXS
can function on one or more remote or local hosts. The number of host machines can vary in time. This indicates that the MAXS platform can adapt to existing situation. In the current development stage, MAXS platform can interact simultaneously with various different OPC DA [14] and database servers obtaining, storing, and processing real time data. In addition to the JADE FIPA [21] specific and OPC ontology based communication, MAXS platform agents can simultaneously reuse integrated databases as a fully functional, fully qualified, parallel communication channel, if needed. Such redundancy can be very helpful - whenever one channel fails unexpectedly, the communication process can still occur by means of the secondary channel without having the problem of either data integrity or performance degeneration. Normally, MAXS reuses the database communication channel to retrieve, store or update both raw process data obtained from the system and platform configuration settings that can be reused during the next system startup. Detailed description about MAXS communication can be found in [54][55].

Currently, in software engineering, implementations of a database management mechanisms are based on the Object Relation Mapping (ORM). Moreover, such an approach fits well with the presented ontology based MAS DCS integration system engineering. This is true, because it is a most suitable, convenient and straightforward approach that improves obtained solution interoperability and interconnectivity. ORM can translate software’s system object model of a different classes and properties into a database's system relational model of tables, columns and keys. It is a programming technique that allows for the conversion between various incompatible type systems, creating as a result a virtual object database that is easily accessible for each software system regardless of the programming language. ORM based database management mechanisms allows to interact with a different database systems provided by a different vendors. This is because ORM promotes reusability, assuring that one database management implementation is capable to interact with each such database system. In the presented ontological MAS DCS integration system cooperation with various database servers is achieved by means of the NHibernate[56][57]. The NHibernate framework, however, is designed to be used over the .Net platform only, and in order to utilize it the MAXS platform was additionally integrated with the .Net platform by means of JADE Sharp [58]. JADE Sharp is a JADE add-on which comes as an additional .dll module. It enables creation of .Net agents compliant with the JADE framework. The original version of JADE Sharp was proposed by TILab, a company which also created the JADE framework itself. However, for the purpose of the proposed MAS, the JADE Sharp add-on module was strongly modified and refactored to fit the needs of the MAXS platform. By means of the modified JADE Sharp module, the MAXS platform can integrate not only with the Java platform but with the .Net platform as well. In order to establish efficient cross-platform communication, each MAXS agent reuses a common XML based messaging mechanism.

The MAXS platform establishes efficient cooperation with various different OPC Servers through JEasyOpc[59]. JEasyOpc is a software tool containing two frontends. The first one
is both Java Native Interface (JNI)[60][61] and Delphi based library providing few public
methods, which allow for a bidirectional communication with OPC DA servers. The second one
is a Java based library fully compatible with the Delphi library module. The Java based part
exposes the public API provided by the Delphi library directly into MAXS. Both the Java
and the Delphi part of JEasyOpc have been strongly refactored and modified to improve their
functionality, runtime stability and performance during concurrent cooperation with different
OPC DA servers.

Figure 7 MAS based DCS integration – reused software development toolset

14. Multi – Agent Infrastructure for the Ontology Utilization

MASs and DCSs share similar characteristics: they are dynamic, decentralized and redundant
solutions [54]. Moreover, hierarchical, multilayered, MAS serving as a DCS’s integration
infrastructure is capable to meet its challenging requirements by dynamically adjusting
to the existing information structure, if properly engineered [55]. MAS is a highly efficient,
scalable and robust methodology utilizing assorted agent and integration system based
techniques (such as reasoning, mobility, cloning, concurrency, subscriptions, autonomous
computing) to support interconnectivity and interoperability during DCS integration and runtime
[54].

MAS, similarly to Service-Oriented Architecture (SOA) [62] is an infrastructure
for organizing and building software systems which are based on interactions amongst various,
loosely coupled and autonomous components. Each such component exposes its functionality
through an ontology based contracts composed of ontological messages. The difference between
SOA and MAS is that in SOA a software component is referred to as a service. In MAS this
software component is referred to as an agent. Another difference between SOA and MAS
is the weight of such a component. In SOA services are usually complex, in MAS agents
are usually simple and lightweight.

MASs proved to be a very promising integration infrastructure, especially because of their
semi-independency, interconnectivity, interoperability and concurrency characteristics [54].
Each such characteristic is a high-level description of a complex functionality. Its delivery is a very time-consuming, hard and error-prone engineering work. MAS, on the other hand, is the appropriate infrastructure by means of which ontologies supporting the notions of interconnectivity, interoperability and semi-independency can be executed. Because of that ontologies can significantly ease up the delivery of functionalities by filling the gap between abstract, high-level feature description and its real implementation. Those features are very important during communication and cooperation activities as they are key features of a MAS based problem solving strategy [54][63]. Additionally, within the commonly used communication protocols, in the DCS, it is necessary to know the data structure provided by each individual subsystem. In MAS, this knowledge is enclosed in ontology [64]Figure 8. MAS by means of ontologies relieves to some extent from time consuming, difficult analysis and decision making processes, making such systems less error prone [54]. Ontology based MAS introduces openness, scalability and the maintainability into DCS integration processes showing how complex and distributed systems can be integrated to achieve quality and great performance [63].

![Ontology Modelling](image)

Figure 8  OPC Server structure ontology modelling

Ontologies introduces flexibility into engineering of MAS DCS integration system allowing it to be structured of many different agents. Each such agent can be written using different programing language. Such an approach allows reusing different Operating Systems (OS) during integration. However, ontologies are the subject of intensive and iterative analysis and validation as well, since similarly to everything else people create, it is impossible to create them from
scratch without any flaw or error. Thus, they have to be thoroughly, effectively and analytically verified. One possibility to perform such analysis and validation is through First Order Logic (FOL). A sample of such analysis which is based on UML as well is presented in [37]. Details of FOL analysis is well described in [65] and [66] as well as in [67]. Furthermore, additional FOL based analysis of an agent ontologies is introduced in [68]. In the proposed approach, FOL analysis is not limited to ontology validation only, as it is reused in a broader aspect during ontological MAS DCS integration system analysis. However, because FOL is a source of undecidability it has to be extended by means of Modal Logic. Such an approach to an ontology analysis allows to study it in terms of fixed set of worlds or states.

Currently, in the field of the DCS integration engineering no attempt has been made towards full utilization of an ontology, FOL and MAS notions during system requirements analysis, design, validation, implementation, maintenance, documentation, runtime data processing, and communication, which soundly stresses scientific contribution of this work. In the presented approach to DCS integration, OPC (Open Platform Communication) specification becomes one of the foundations of the ontology establishing an easy way to present and exchange various pieces of the process control information. The established ontology serves as a medium for domain knowledge analysis, a solution model, an implementation and documentation base, and a communication medium. It gathers and organizes all required, significant pieces of information about underlying DCS, reducing its integration system complexity by improving interconnectivity and interoperability of integration system sub-components. It is reused during database integration serving as a reference towards database schema creation; during engineering of MAS DCS integration system, serving as a solution model; during documentation and implementation reference towards agents’ specification; during runtime serving as a communication medium between different agents. Finally, it is worth mentioning that the obtained DCS integration system is not only ontology based as it is MAS based as well.

15. MAS Infrastructure for the DCS Integration

The MAXS platform in the current development stage consists of seven different types of agents organized hierarchically on three different logical layers, integrating given DCS Figure 9 and Figure 10. However, it remains open for the further development as it is a flexible and scalable solution. MAXS agents are: 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 [63]. In short, SA is used to dynamically administer other platform agents that show up 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 OA pool of agents. The number of created OA agents reflects the number
of discovered OPC Servers. 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 process supervision and control through hierarchically structured, automatically
synchronized OPC groups and items tree.

Figure 9 MAXS layered structure – communication types

The number of MA agents varies in 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 a 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 direct agent communication channel, for unknown reason,
will be unavailable. NA is responsible for manual remote host registration to the existing MAXS
platform. A complete description of the platform architecture and each platform agent
was presented in [54] and [55]. 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 [21].
16. Conclusions

This work focuses mainly on the notions of interconnectivity and interoperability that exists amongst different software based subsystems which forms a greater integration system. Interconnectivity and interoperability issues, viewed from the analytical and practical perspective, shows how DCS integration problems can be solved by means of existing software development and modelling tools. From the authors’ perspective, ontological MAS, is appropriate software infrastructure capable of handling with each problem related to the DCS integration on the different stages of integration system manufacturing such as design, implementation and maintenance. Moreover, this work contributes to the problem of efficient ontology engineering, focusing on the reusability of a contemporary software engineering tools. Although there are some specialized tools for the ontology development, the authors decided to check if it is possible to reuse basic software modelling and development tools during ontology based MAS DCS integration system engineering. This is because of the fact that those tools are currently heavily reused in modern software engineering by professionals. Moreover, such approach does not favor any vendor specific language and technology and remains open to reusability of current technology investments, flattening learning curve by leveraging widely adopted toolchain for software design and construction. Obtained results indicate that such an approach is correct. This is true because it is possible to compose good quality, highly compatible, and low-cost software modelling and development environment for the ontology
based software development, following at the same time best practices derived directly from software engineering.

Equally important to a domain integration system development methodology is a toolset supporting its engineering, used also during problem solving processes. This is true because even the best methodology without proper toolset support can spawn completely unfortunate solutions. It is a responsibility of each engineering team to choose between the best and most reliable development tools available.

Ontology based MAS methodology has been known for quite some time, and in the current situation can be already supported by a wide variety of good quality, free of charge or commercial development tools, which makes this approach a straightforward undertaking, at least from the development point of view. The only problem that needs to be answered 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 contents of such a toolset, presenting the meaning and the usage of each particular element giving birth to MAXS – a functional ontology based MAS – which is capable of integration of the dynamic DCS composed of various sources of data.

In the presented approach to DCS integration system engineering, UML, XSD and source code are interchangeable and of equal support during creation of an ontology representing the 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 different ontology formats. The UML class and package diagrams are used as a formal domain ontology model. XSD serves as ontology machine readable format, and during runtime it is a backbone of a data validation facility. XML documents are instantiation of an XSD and are reused during messaging processes between various different MAXS platform agents. Both UML and XSD are interchangeable formats in the aspect of the automatic ontological source code generation. Moreover, XSD strongly supports portability of the obtained ontological source code between different programming platforms, such as in the case of Java and .Net. Solution specific source code is designed to reuse both XSD and XML during runtime. It is a machine executable ontological format. It is generated automatically and then customized manually according to the planned functionality. FOL is an analytic tool allowing to validate not only the ontology structure but ontology based algorithms as well. To overcome the problem of FOL undecidability presented FOL has been extended by means of a Modal Logic. Such an approach to an ontology analysis allows to study it in terms of fixed set of worlds (states). Such an approach enables creation of domain based ontology maintaining this way, from the bottom, the data integrity level, reusing well known formats of various domain data structures.

Presented OPC based ontology, is a result of such an approach. None of this actually would be possible if there was not to be a reasonable solution for the problem of software IDE
configuration supporting in each development process activity starting from the design and ending on the implementation activities. Each ontology, including the created OPC based one, can be utilized by means of the properly engineered MAS infrastructure. This is true because MAS composed of various agents can be designed according to the different parts of an ontology. MAXS platform is a working example of such an infrastructure.

The presented approach utilizes MAS to process ontology which has been, a well-known idea for some time. However, this well-known idea, for unknown reasons never got the real chance to move from the academic circles to the real life, to help solve various complex problems. That is why the idea of ontology based MAS integration is quite new in many areas. This seems even more surprising, given that ontology and MAS, similarly to the traditional solutions, derive from the best OO practices, providing enough capabilities to solve a variety of domain-specific integration problems. This alone is a great contribution of the ontology based MAS to the field of problem solving strategies for a various kinds of businesses.
References


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


[29] Belardinelli, F., Lomuscio, A., "Quantified epistemic logics for reasoning about knowledge in multi-


