Towards the deep, knowledge-based interoperability of applications

Andrius Valatavičius

Vilniaus universiteto doktorantas
Vilnius University, Doctoral student
El. paštas: andrius.valatavicius@mii.vu.lt

Saulius Gudas

Vilniaus universiteto vyriausiasis
mokslo darbuotojas, daktaras
Vilnius University, Doctor
El. paštas: saulius.gudas@mii.vu.lt

The interoperability of enterprise applications in a dynamic environment is a complex issue. New methodological approaches and solutions are required. The methodological background of our approach is the internal modeling paradigm integrated with MDA approach. The modified MDA schema includes the new layer of the domain knowledge discovery, frameworks for internal modeling of enterprises. The peculiarity of the modified MDA is a focus on the cross-layer transferring of domain causality. The presented frameworks will help to trace the domain causal dependencies across the layers of the software system development, and they will aid in determining the influence of domain causality to the integrity and interoperability of the application. Researchers consider that the dynamic enterprise domain must be a goal-driven and self-managed system. The management transaction concept uses the internal modeling of the enterprise, which reveals the goal-driven information transformations inside the enterprise management activity (deep knowledge). This approach is combining the business process modeling and control theory principles, enterprise architecture modeling and autonomic computing concepts. The ArchiMate enterprise architecture modeling language is used for illustrating the cross-layer transferring of domain causality. Finally, we developed the architecture of the interoperable enterprise applications with the autonomic integration component.

Keywords: *internal modeling, enterprise management, domain modeling, self-managed system, MDA, knowledge discovery, interoperability, autonomic computing.*

1. Introduction

The interoperability of applications in a dynamic enterprise environment is a complex issue. In this article, we present the methodology for maintaining the interoperability of the applications using autonomic computing and business process models. In the constant growth of enterprise complexity, more various applications are used in a single enterprise (e.g., accounting systems, CRM, ERP, and E-Commerce applications), data integration and application interoperability become pressing problems for technological advancement. Currently, the integration of the applications is expensive, and projects mostly tend to fail (Halevy et al. 2006; Trotta 2003; Valatavicius et al. 2014; van der Bosch et al. 2010). Multiple conflicts may occur in the data integration process (Dong et al. 2009). This article deals with enterprise interoperability and aims for an integrated information system design. Five problems of software system interoperability arise in a dynamic business environment. First, Applications (i.e., in number or provider) are changing over time in a dynamic enterprise domain. Second, it is usually more than one application in an organization environment and the number may vary over time, causing demand for data migration project development. Third, there are no common methods to describe the collaboration among multiple different dynamic applications. Fourth, when the software changes (i.e., when it is updated or switched to software from different manufacturers), the business process might also change adapting to the new requirements of the environment, then the static business architecture model becomes invalid. Fifth, to ensure interoperability, the integration expert needs to perform the following tasks: perform the schema alignment (Hophe et al., 2002; McCann et al., 2005; Peukert et al., 2012; Rahm et al., 2001; Silverston et al., 1997); ensure record linkage and data fusion (Dong et al., 2013; Kutsche, 2008); ensure the orchestration and choreography of application services and data objects. In a dynamic environment, business processes often need optimizing, akin to the El-Halwagi examples of business process integration (El-Halwagi, 2006; Pavlin et al., 2010).

Various application integration methods are applied to maintain the interoperability of enterprise applications. Most researchers of integration subject use advanced methods, such as agent technologies (Overeinder et al. 2008) and ontology-based technologies (Li et al. 2005; Shvaiko et al., 2013). We notice that the enterprise architecture frameworks (MODAF, NAF, DoDAF, TOGAF, GERAM) are a good way to represent of real-world processes (i.e., capture the business domain knowledge), and thus their interfaces with applications layer components. Software developers rarely explore business processes for the application integration solutions. However, sophisticated methods of the process integration already exist (El-Halwagi 2006) – they're just not being applied in the application area.

This paper offers the internal modeling paradigm consolidation with the Model Driven Architecture approach (OMG MDA). The MDA approach is modified and presented to illustrate the qualitative differences of the software engineering in the internal modeling paradigm. The theoretical background of the presented approach is starting from the regulator theorem (Conant et al., 1970). We continue with R. Ashby conclusions 6/18 of the assembly of Black Boxes and "emergent" properties (Ashby 1957), the definitions of second order cybernetics (Heylighen et al., 2001) and the autonomic agents and autonomic computing (Kephart et al. 2003). The main principles of data integration and engineering solutions are refined using the ArchiMate enterprise architecture language (ArchiMate, 2016). This research would help to work out the methods to support analysis of the business domain and enterprise software collaboration processes.

Business domain knowledge, acquired from all the available sources, can be of benefit to support the application integration solutions. Business domain modeling itself is a complex problem, for which it is required to solve another complex issue.

Model-driven software development (associated with MDA, BPMN, DMN), the enterprise architecture modeling frameworks (e.g., ArchiMate, MODAF, NAF) are based on business domain modeling and are aimed at the development of the software systems. The complex software systems are often implemented using the agent technologies. For instance, a background for using the Platform Independent Models (PIM) for autonomic agents development is presented in a study by Zinnikus et al. (2008).

The dynamic nature of the business processes causes many problems with the already developed enterprise architecture and business process models, as well as with the implemented (legacy) applications. The most common scenario is when changes in business force to replace the outdated legacy software by one or multiple new software items, which are designed for some specific business process (i.e., bookkeeping software, enterprise resource planning system or e-commerce software).

Meanwhile, a lack of focus to the complexity of business domain in the information systems engineering methods (including enterprise modeling, business process (BP) modeling, enterprise software design) slow down the enterprise software adjustment to environment changes. In our approach, a business domain (an enterprise) is considered as a complex system: a dynamic, goal-driven and self-managed system, formally defined as an organizational system (Gudas 2012a; 2012b). The definition of management transaction is the base of the internal enterprise model, which acquires the essential causal dependencies of the domain – goal-driven information transformations inside the management transactions (deep knowledge). The principles of the second order cybernetics provide the methodological basis for the internal viewpoint (Heylighen et al. 2001), and they aim to disclose the internal causal relationships of the domain. Internal modeling seeks to construct a white box model of the domain, while other methods of enterprise and business process modeling (DFD, BPMN, IDEF, ARIS and others) examine the domain using the external modeling paradigm as a structure of black boxes, for example, as a set of workflows (Input, Process, Output) or as an event-process chain. We applied the constructive research method, which is aimed to reveal domain causality and to determine the impact on the integrity and interoperability of the application, by applying the systems analysis, control theory principles and using enterprise architecture modeling and autonomic computing concepts. The methodological background of our approach is the modified MDA schema, which includes the new layer of the domain knowledge discovery.

The combination of the disciplines of control theory, business process modeling, enterprise architecture modeling and autonomic computing concepts allows us to reconsider the model-driven development aspects. Our approach is a consistent realization of the internal modeling paradigm integrated with an MDA approach. The presented enterprise modeling frameworks are focused on acquiring the essential causal dependencies (deep knowledge), paying attention to the content of the enterprise management transactions.

One of the research questions we ask is whether internal modeling with an MDA approach helps to determine the influence of domain causality to the integrity and interoperability of applications. Second, is it possible to create an architecture of autonomous interoperable enterprise applications using only business process models and an enterprise architecture model?

The preceding discussion implies that the software systems and enterprise management activities are aimed to adopt business environment changes in a similar way to the classical control system with a feedback loop. The idea is to adopt the internal model control principle defined by the good regulator theorem (Conant et al., 1970) for enhancing the intelligent software technologies (e.g., intelligent agents, autonomic computing components). According to the good regulator theorem (Conant et al., 1970), the Internal Model (IM) is a predefined knowledge structure, based on the essential properties of the particular type of domain. Thus, an internal modeling paradigm entails the usage of an essential (deep) knowledge of the enterprise domain. The causal dependencies inside and between the enterprise management activities are considered as essential (deep) knowledge. The internal structure of the enterprise management activity is defined as a management transaction (MT) (Gudas et al. 2016), and, on the detailed level, it is defined as an elementary management cycle (EMC) in studies by Gudas (2012a; 2012b). Our paper contributes to the theory of application interoperability by proposing an inter-dimension approach of multiple integration levels (Technical, Semantic/Data and Organization), which are defined in the European Interoperability Framework (EIF) and mentioned in multiple other articles (EIF 2004; F.B. Vernadat 2007).

This manuscript is structured as follows. In the first section, we present the technological background supporting our software integration approaches processes. We describe the internal model-based control system and the core of the good regulator theorem in Section No. 2. The external and internal modeling paradigms in the software engineering are elucidated in Section No. 3. Section No. 4 includes the modified MDA schema with two modeling paradigms, assumptions of the internal modeling based enterprise software development, and it illustrates the internal modeling of enterprise domain using ArchiMate. In the same section, we discussed the enterprise management modeling frameworks and autonomic computing technology. Section No. 5 is dedicated to the application of the interoperability problem using an internal model and describes the architecture of the enterprise applications with the autonomic interoperability component. The sixth section introduces the prototype of the software interoperability validation tool. The results and further work required discussed in the concluding part.

2. The Core of the Internal Model-Based Control System

The internal model was defined in 1970 as a good regulator theorem (Conant et al. 1970). The regulator theorem is the following idea: "any regulator (if it conforms to the qualifications given) must model what it regulates" (Conant et al. 1970). Internal modeling can be used for the enhancement of intelligent software technologies (i.e., utility-based or intelligent software agents) and serve as a background of knowledge-based software systems. The internal model first was articulated as the internal model principle of control theory in 1976 (Francis et al. 1976). The internal model approach emerged in the control theory, the problem of the domain of which is a device, object or an open system in general (Fig. No. 1). The purpose of the Internal Model is to supply the closed loop of the Control System with control signals which maintain the stable behavior. The Internal Model (IM) is a model of the problem domain (in this case, IM is the model of System S). The Internal Model is created in advance using prior knowledge, i.e., IM is a predefined model, based on knowledge of the essential properties of the domain. In other words, IM is a knowledge-based model of the controlled system S, integrated within a Control System. Due to IM, an important, intelligent feature of prediction occurs in IM control systems because the control is based not only on the measurements or evaluation of the state (Kumar 2012).

Control System and Systems S. Here, *u* is a command to control action, *y* – the measurements (system state attributes), *r* is the reference input and *d* is the disturbance signal. A second feedback loop is an internal transfer of information flows (command (u), system state attributes (y), internal feedback (d~)) between Controller C and Internal Model, where d~ is an internal feedback flow. Internal model control processes the system state measurements (system attributes (y)) and compares them to Internal Model output (prediction).

The Control System, with the internal model, is adaptive to changes in the environment. Therefore, we are convinced that the adoption of the the internal modeling approach for the software system interoperability looks like a promising and novel solution. Researchers have already applied principles of control theory in software development (i.e., intelligent agent technologies, autonomic computing), but the employment of the internal models for application interoperability are arguably rarely occurring.

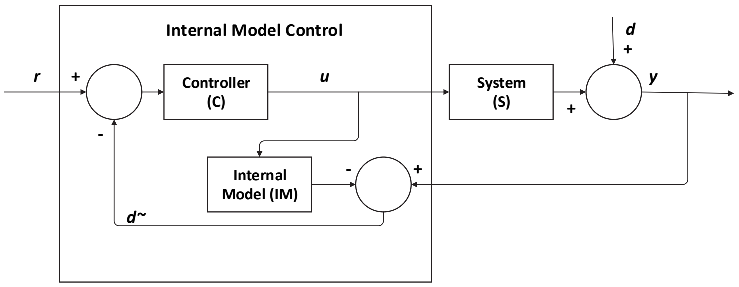


Figure No. 1. ***The internal model-based control system.***

It is important to note that the IM in control theory (Fig. No. 1) contains a model of the essential causal dependencies of the domain inside the Control System (the perceived causality assuming that IM = S). These models are based on models created by MDA approach; therefore, in a dynamic enterprise environment, we have models that are always up to date. The content of the feedback loop between elements of the system is the transmission and processing of data (signals) flow, so it should be called a transaction. It is worth mentioning that the internal model-based control system includes more than one feedback loop. The external feedback loop transfers information flows (command (u), disturbance (d), system state attributes (y)) between

The role of the internal model in control theory and the role of the domain model in the knowledge-based software engineering – both approaches are well compatible with each other, because they are relevant to the principle of internal modeling (Fig. No. 1) (Gudas 2016). In general, the internal modeling focus on the discovering of deep knowledge of the problem domain, i.e., the internal modeling is aimed to reveal causal dependencies of the problem domain.

Examples of usage of the internal modeling approach (quality management, risk management, business process management) (Moen et al. 2006; Brache 2002) are provided in frameworks: Fayol's business function model (Fayol 2016); Deming's PDCA cycle; Porter's Value Chain Model (Porter et al. 1985); Rummler-Brach's enterprise performance management model.

The concepts of control theory have a need for controlling interactions of enterprise software components and the integration of applications. Researchers apply control theory in multiple fields, e.g., intelligent agents, autonomic computing (Kephart et al. 2003), reactive software applications (Winter et al. 1998), adaptive systems (Mareels et al. 1996) and computing systems (Abdelzaher et al. 2008).

Considering the internal model (IM) as a knowledge model of the problem domain inside the control system used to maintain the stable behavior. IM is also considered as a white-box of the problem domain (A Controlled system S in Fig. No. 1), which specifies the essential elements and their dependencies of the problem domain (laws of behavior inside a domain).

Analysis of the role of the internal model (IM) in control systems allows concluding that the adaptation of the internal model (IM) in the context of software systems development is a relevant topic for enhancing intelligent technologies.

The architecture of the intelligent software components with the internal model (e.g., intelligent agents or autonomic components) in Fig. No. 2 is relevant to the structure of the internal model control system (Fig. No. 1). The similarity of real world RW process control systems and the intelligent software systems is evident if both systems include the internal models of the subject domain. The internal model in relation to MDA processes models is created using MDA approach in M in Internal Model of Domain in the IMC-based item of each software system (Fig. No. 2).

3. Two Modeling Paradigms in Model-Driven Software Engineering

At present in software engineering external modeling paradigm is prevailing, because of black box approach modeling usage in various SDLC phases. In science and engineering, a black box is a device or systems which the inputs and outputs without any knowledge of its internal structure and processes.

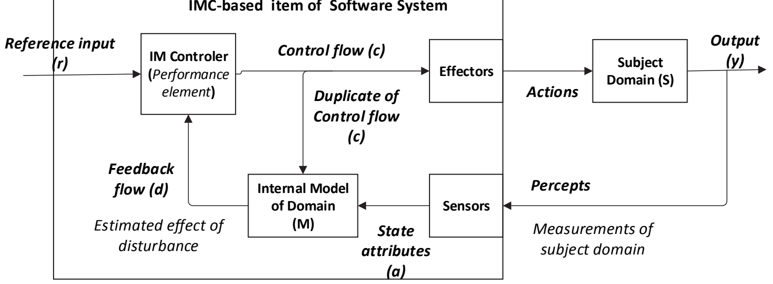

IMC-based item of Software System
Reference input (r)
IM Controler (Performance element)
Control flow (c)
Effectors
Actions
Subject Domain (S)
Output (y)
Duplicate of Control flow (c)
Feedback flow (d)
Internal Model of Domain (M)
Sensors
Percepts
Estimated effect of disturbance
State attributes (a)
Measurements of subject domain

*Figure No. 2. **The architecture of a software component with the internal model.***

Enterprise information system (software) engineering methods include business domain modeling languages (e.g., BPMN, DMN, UPDM, UEML), enterprise architecture frameworks (e.g., DoDAF, MODAF, ARIS) and software design languages (e.g., UML, SysML, UPDM). These languages and frameworks are used to construct the project models required by the corresponding SDLC phase, which is essentially an assembly (hierarchy) of black boxes (Input, Output) with the identifier (name). It is important to note that such methods do not seek to reveal the domain causation; it is enough to describe the externally monitored interactions.

The business process modeling, business activity modeling are the source of knowledge for software development solutions, and, arguably, integration experts can use those for supporting the interoperability of business management applications (e.g., ERP, CRM, E-commerce, accounting systems, collaborative software).

The concept of "an internal model" is covering a range of models, which are developed using prior knowledge of problem domain, i.e., an internal model is relevant to the grey-box and white-box models. Multiple domains apply to the Internal models, i.e., in medicine and biology theories of visual perception, brain functioning, and the motor control system of the body (Francis et al. 1976) underlie an ability to control the unknown and underdetermined changes in the environment. Important is the usage of the Internal models in the business management domain (e.g., risk management, capital management), whereas this domain is an organizational system, the same type of complex systems as well as in enterprise software engineering. The necessity of the internal modeling for acquiring a deep knowledge is confirmed by the R. Ashby conclusions 6/18 of the assembly of black boxes and "emergent" properties (Ashby 1957, 110): "Thus an assembly of Black Boxes, in these conditions, will show no 'emergent' properties; i.e. no properties that could not have been predicted from knowledge of the parts and their couplings."

The level of awareness of the real-world domain (i.e., the level of a prior knowledge) is increasing when moving from black-box models toward a grey-box and, finally, a white-box model (Fig. No. 4). The signifi-

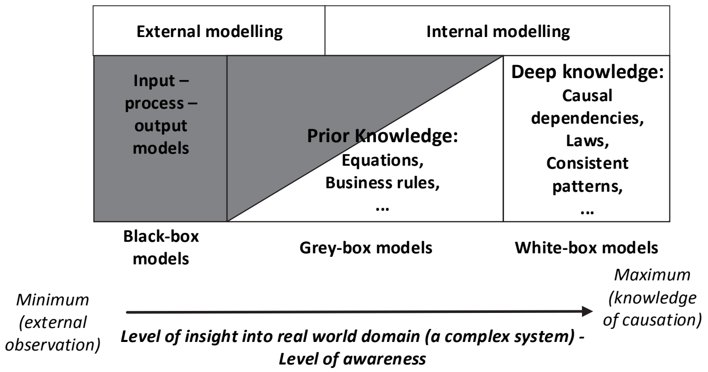


Figure No. 3. ***The significance of the white-box/grey-box models is a depth of insight into the problem domain.***

cance of internal models is the depth of insight into the complex systems (problem domain). The depth of knowledge is increased sequentially in the transition from the grey-box models to the white-box models (a white-box concept marks a maximum level of insight). The internal modeling paradigm introduced in the model-driven software engineering with the intention to enhance the knowledge-based software development methods. In relation to MDA, its modeling techniques can be considered as gray box modeling in our specific scenario, and that still gives us additional information for the interoperability solutions on application structure and about the interrelations of the application via business process modeling. The better the MDA models are, the deeper the knowledge there is, and the better the Internal model controller performs with the autonomic functions.

4. External and Internal Modeling in the Enterprise Software Engineering

4.1. Model-Driven Development and Two Modeling Paradigms

The usage of the internal modeling in the MDA for intelligent software development and deep knowledge discovery (the elicitation of the internal model of the enterprise) are two corelated issues.

Fig. No. 5 illustrates the role of the internal model in the MDA approach. We accept that transitions between two layers of MDA are possible only because of a role of IM: a higher-level IM is used to control the transformation between layers, and to get a content of the mode on the lower layer. The role of the internal models IM(1) – IM(4) in the transformations between MDA layers is twofold. Primarily, the internal models are a part of the "awareness" of the staff (e.g., the business analyst, the architect) used for the development of solutions on the corresponding layer. Second, in the case of software-based mapping between the MDA layers, the internal models could serve as key elements of intelligent (or autonomic) software components (agents).

The additional layer of Real World (RW) domain is added to depict the domain knowledge elicitation step. The mapping of the RW domain to CIM layer models is defined as domain knowledge discovery (knowledge elicitation, Fig. No. 5). Domain modeling reveals that the adequacy of the follow-up project solution directly depends on the "deepness" of domain modeling, i.e., it depends on the capabilities of the knowledge elicitation methods. The CIM layer content adequacy to RW domain properties (the validity of CIM content) depends on the modeling paradigm, as discussed above. So, the advantage is on the side of the internal modeling paradigm-based methods. For instance, the OMG reference for CIM layer modeling is BPMN, which represents an external paradigm based language: the BPMN diagrams are (input, output) descriptions of real-world processes (black boxes) and can't be called specifications of domain causality. Meanwhile, the new OMG specification DMN (Decision Modeling Notation) is an example of language, based on the internal modeling paradigm. DMN is a new step in RW domain gray box modeling, because it is focused on the domain internal dependencies – business rules modeling (Kardoš et al. 2010).

Some workflow modeling methodologies attempt to model the domain causality, i.e., internal dependencies modeling, e.g., the ActionWorkflow Approach, Workflow Management (communication-based work-

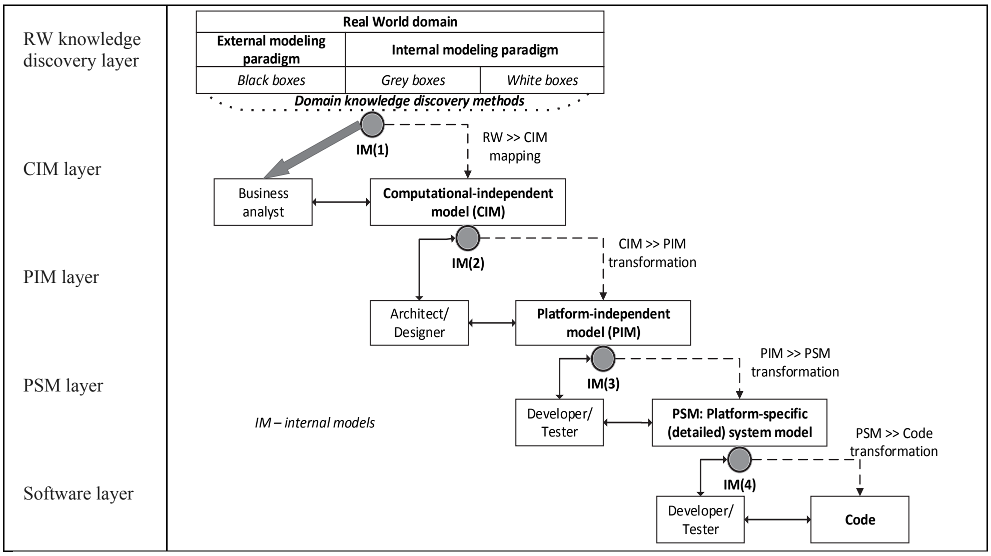

RW knowledge discovery layer
CIM layer
PIM layer
PSM layer
Software layer
Real World domain
External modeling paradigm
Internal modeling paradigm
Black boxes
Grey boxes
White boxes
Domain knowledge discovery methods
IM(1)
RW >> CIM mapping
Business analyst
Computational-independent model (CIM)
IM(2)
CIM >> PIM transformation
Architect/ Designer
Platform-independent model (PIM)
IM(3)
PIM >> PSM transformation
Developer/ Tester
PSM: Platform-specific (detailed) system model
IM – internal models
PSM >> Code transformation
IM(4)
Developer/ Tester
Code

Figure No. 4. ***The modified MDA schema includes two modeling paradigms.***

flows, Winograd et al. 1987; Medina-Mora et al. 1992), and the transactional workflows (Georgakopoulos et al. 1995).

An example of the internal domain modeling from the functional perspective is that the domain modeling method developed by Osis (2004). The Functioning Cycle in the mentioned study (Osis 2004) is a key construct of the domain, a form of cause-and-effect relations modeling for the software engineering needs.

Some enterprise software development methodologies are based on the domain-related theory (e.g., meta-models, ontologies). Some are aimed to capture deep knowledge while exploring the domain meta-models: UEML – the Unified Enterprise Modeling Language (Vernadat 2002), EEML – the Extended Enterprise Modeling Language (Krogstie 2005); enterprise domain ontologies (Zachman et al. 1987; Dietz 2006).

The modified MDA approach in Fig. No. 4 includes two modeling paradigms: external and internal. The external modeling paradigm is explored by traditional software development methods, when software development begins on a CIM layer using BPMN (e.g., IDEF, DFD) to represent the external observations of domain activities, i.e., to omit the domain knowledge discovery (based on some theory of domain). The internal modeling paradigm is theoretically based on the good regulator theorem (Conant et al. 1970; Francis et al., 1976). Further, this manuscript presents the internal modeling based technique to maintain the interoperability of applications. Internal modeling can be a basis for the enhancement of the knowledge-based software development methods.

4.2 Assumptions of the Development of Internal Modeling Based Enterprise Software

This approach of internal modeling in enterprise software engineering is based on the assumptions as follow:

Assumption No. 1. The knowledge-based software development methods should be deep knowledge-oriented, i.e., based on the domain causal dependencies discovery, and this is the internal modeling paradigm.

Assumption No. 2. The modified MDA approach (Fig. No. 4) includes the knowledge discovery layer, and is defined as the sequence of cross-layer transformations based on the internal model control principle:

$$IM(1) \rightarrow IM(2) \rightarrow IM(3) \rightarrow IM(4), \quad (1)$$

here IM(1) is a domain knowledge model (DKM), IM(2) is an enterprise/business process model (CIM), IM(3) is a software system architecture (PIM), IM(4) is a detailed software system model (PSM).

Assumption No. 3. The essential features of the real world domain, which accumulate in the internal model IM(1), must remain in the lower layers of MDA, i.e., they should transfer (transform and remain) in the internal models IM(2), IM(3) and IM(4). The extended model of the MDA approach (Fig. No. 4) includes two modeling paradigms: external and internal modeling (see Fig. No. 1). Thus arises the second dimension in the MDA, which evaluates the validity (accuracy) of modeling, i.e., the depth of the obtained knowledge on the CIM, PIM and PSM layers. The matter of using the internal modeling paradigm is to acquire the essential (deep) knowledge of the subject domain for software development needs, while paying attention to the specifics of a domain. In some software technologies (e.g., intelligent agents), the domain knowledge model has been included: the condition-action rules, utility functions, performance elements.

There are two questions concerning the validity (relevance, completeness, accuracy) of IM in the engineering methods of the enterprise software system:

1) Does the IM includes or does not contain any deep knowledge of the domain? The question is of the degree of relevance of IM content against causal dependencies of the real world domain: is IM an external model (a black box), or is IM an internal model (a gray box or a white box model) of the domain causality.
2) Is there enough of IM content for the needs of the software development method? The question is of the relevance of IM content against the particular methodology and the methods of the enterprise software development approach.

▫ Arguments for Assumption No. 1

The assumption one is proven by the analysis of the qualitative differences of the internal modeling and external modeling of the enterprise domain in (Gudas, 2016), (Gudas et al., 2016). We focus the internal modeling paradigm on the deep knowledge seeking to reveal the consistent patterns and dependencies (laws) within the problem domain. Therefore the internal modeling is critical for advancing knowledge-based modeling methods. An enterprise domain perceived as a type of complex systems - an organizational system, a self-managed system with hierarchical structure, goal-driven activities, that transform the data and knowledge and are directed to produce the output of the system. Such understanding of domain properties is in line with the 2nd order-cybernetics viewpoint (Heylighen et al. 2001).

Fig. No. 5 depicts the key elements of the enterprise meta-model described in studies by Gudas (2016; 2012a). Our approach works in with the condition that enterprise management activity, in a real world, is a self-managed system. The management transaction (MT) defines causal depen-

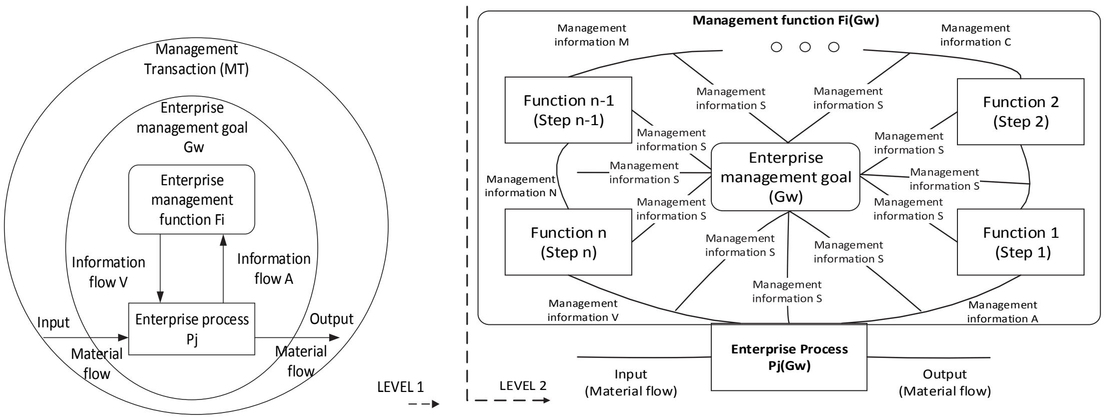


Figure No. 5. ***The knowledge components of the enterprise domain:***

a) The conceptual representation of a management transaction (MT) at level 1) and an elementary management cycle (EMC) at level 2).

dencies inside the enterprise management activity, namely, a feedback loop between management function (Fi) and enterprise process (Pj). The management transaction (MT) is a control view-based content of an enterprise management activity on level 1 in Fig. No. 5a (Gudas 2016). Therefore, an internal model of MT is the elementary management cycle (EMC), which is depicted on level 2 and must also be as a self-managed system. The general internal structure of EMC (Fig. No. 5a) is discussed in the abovementioned studies (Gudas 2016; Gudas 2012a).

An example of the management transaction (MTij) in Porter's Value Chain Model (VCM) is an interaction between primary activity (manufacturing process Pj) and support activity (management function Fi). Fig. No. 5b depicts an example of EMC(i,j), which, adopted for enterprise software engineering needs, is in the BPMN notation. The elements of the EMC (i,j) contains the process (Pj), the goal (G), and the management function (Fi). They, in turn, comprise the goal-driven information transformation steps (IN, DP, DM, and RE), the information flows (Flow1, .., Flow5), the impact of goal (information flow S), and a feedback loop with the process (Pj). So, an example of any deep knowledge of the enterprise domain are these two components – the management transaction (MT) and the Elementary Management Cycle (EMC) – both considered as a self-managed system (Gudas 2016).

▫ Arguments for Assumption No. 2

The second assumption is that the software development as a sequence of internal models mappings IM(1) à IM(2) à IM(3) à IM(4) could be proven using the enterprise architecture development methods. The enterprise architecture (EA) frameworks (e.g., DoDAF, MODAF, NAF) usually include few modeling layers (so called *views* or *viewpoints*) as follows: motivation/strategy view (corresponds to IM1), operation view (corresponds to IM2), system view and service view (corresponds to IM3). The enterprise architecture (EA) development process is based on the mappings between these EA views. Fig. No. 6 presents the OMG MDA approach alignment with the ArchiMate meta model (fragment).

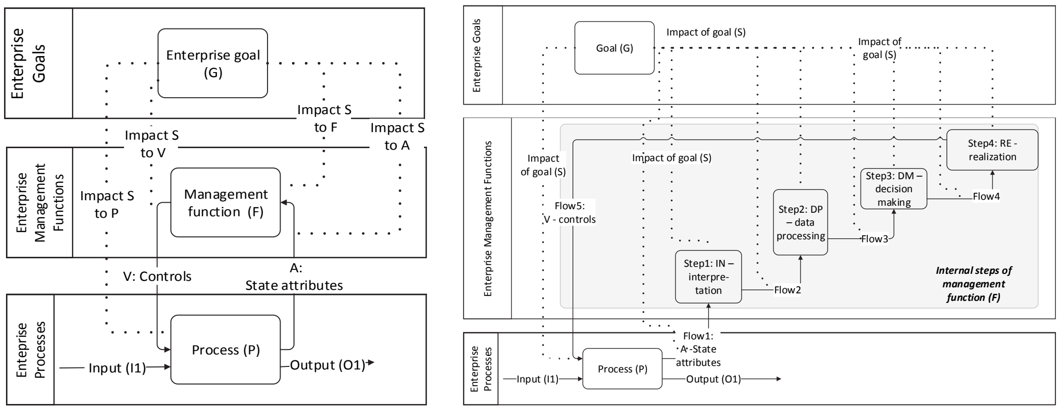


Figure No. ***6. The knowledge components of an enterprise domain:***

b) Adopted for enterprise software engineering MT and EMC frameworks (BPMN notation).

The ArchiMate framework is one of the examples of the external modeling approach (ArchiMate 2016), because here, the key concepts are modeled in layers (e.g., business, application, technology, strategy, and motivation), and cross-layer transformations are based on the mappings of concept to concept. The alignment of the MDA approach (OMG) and ArchiMate in Fig. No. 6 reveal some typical properties of the external modeling:

- Considering IM1: Domain knowledge model is not specified explicitly in the MDA approach. Assumably, the CIM level includes the elicitation of domain knowledge. However, a domain knowledge discovery is an important issue for ensuring system quality; so, it should be specified explicitly. The ArchiMate framework includes RW domain knowledge (IM1), whereas motivation and strategy elements (e.g., goals, drivers, requirements, capabilities) are representing the needs of stakeholder: the motivation element *Goal* realized by the strategy element *Requirement*, which adjusts through the *Capability* element. However, the mapping of IM1 to IM2 is carried out through only one concept of *Capability* to the concept of *Business Service* (or *Business Process* or *Business Function* or *Business Interaction)*.

IM2 corresponds to CIM in MDA, and to the Business layer model in ArchiMate. Both are focused on the business requirements (business logic and rules). However, properties of CIM are not predefined (or constrained) by the meta-model. Therefore, each domain modeling method is appropriate. The recommended one is BPMN, and now even DMN (since 2016). Meta-model predefines IM2 in ArchiMate. Although, it is a concept map, which based on the experience (has no theoretical justification). Considering IM2 in both cases (for MDA and ArchiMate) it can be asserted that a real world domain not perceived as a complex system, i.e., IM2 is not intended to capture the causal dependencies of the domain. So, in both cases, IM2 can be seen as a black-box model (an external modeling paradigm).

- Considering IM3: Internal Model (IM3) corresponds to PIM in MDA approach. IM3 belongs to the application layer in ArchiMate framework and meta-model

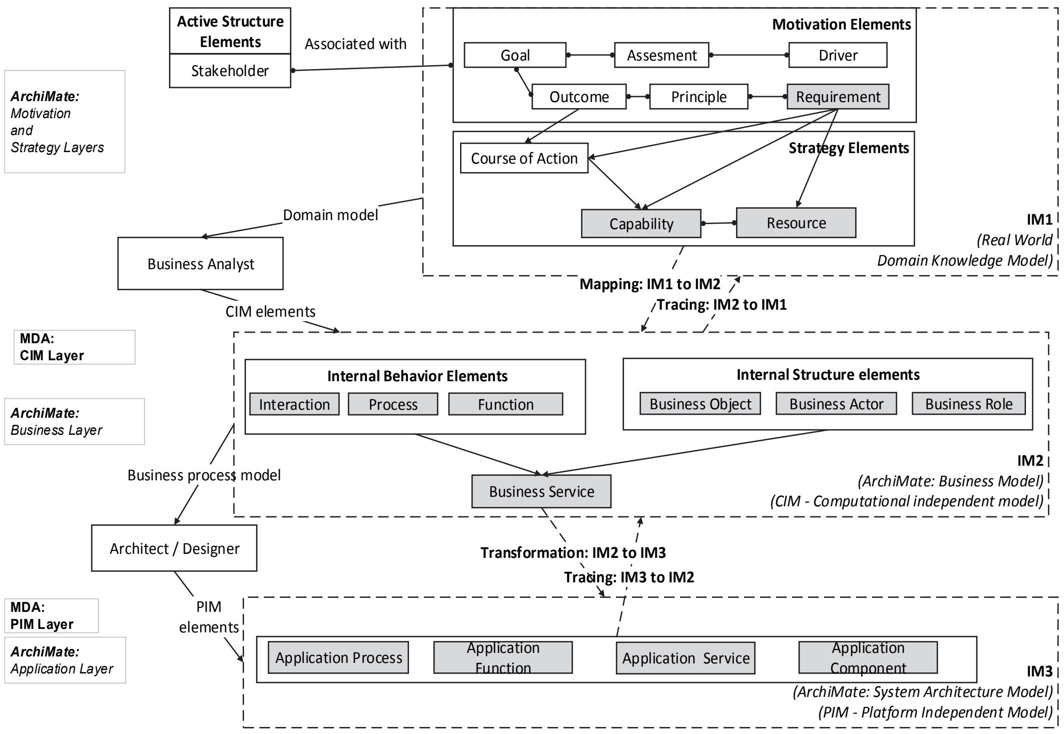


Figure No. 7. ***The OMG MDA approach alignment with the ArchiMate framework.***

defines it. IM3 is a kind of a concept map, based on the experience (has no theoretical justification).

- Considering transformation IM2 to IM3: MDA defines the transformation CIM to PIM as a generalized requirement - the CIM constructs should be traceable to the PIM, and PSM constructs that implement them (and vice-versa). In ArchiMate the transformation IM2 to IM3 is predefined in the meta-model by cross-layer associations of key concepts (Fig. No. 7). In both cases, the mapping of IM2 to IM3 is seen as a mapping between concepts (entities, or objects). So, the transformation IM2 to IM3 is not defined here as the mapping between complex structures, when it includes transference of systems regularities.

▫ Arguments for assumption 3

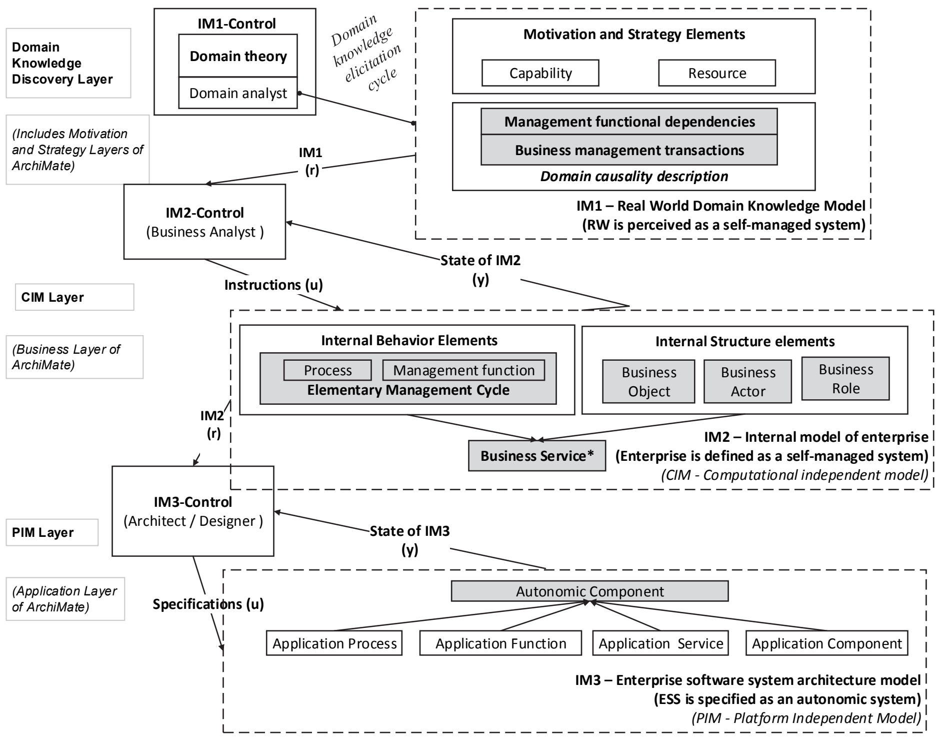
The third assumption is that the transferring of the essential features *of the RW domain on the lower layers of modeling* is proven by the comparison of IM1, IM2, and IM3 internal structures (Fig. No. 8).

First of all, the relevance of the domain knowledge discovery method to the type of the domain is an issue. This issue is a fundamental issue of the domain modeling, which determines the relevance (validity) of IM1 against the RW domain: are the causal dependencies captured in IM1 enough for research purposes or not? From the internal modeling perspective, a theoretical background (domain theory) is required for recognizing essential features of the domain by domain analyst (presented as IM1-Control in Fig. No. 8).

The second issue focuses on the internal modeling requirements for the MDA cross-layers interactions. The internal modeling paradigm using in the modified MDA (Fig. No. 4) requires maintaining the essential feature of the IMC system discussed above (Fig. No. 1). Namely, the IMC system consists of two components: Controller (C), which forms a control solution, and the Internal Model (IM), which is the inner knowledge of the IMC and is correlated with the content of a particular layer.

In the present case, the mapping of captured RW knowledge (IM1) to the lower layer (IM2) is under control of IM2-Control component. The mapping IM1 to IM2 is performed by a business analyst or software tool in a way when substantial causal dependencies (fixed in IM1) transferred to the lower layer model IM2. It is evident that the output of the IM2-Control is IM2. The content of IM2 depends on the input (IM1), and on the internal knowledge model IM2* of the IM2-Control component, required to control IM1 mapping to IM2. We notice that a business analyst or software tool should perform a transformation of IM2 to IM3. The content of IM3 depends on the input (IM2) and on the internal knowledge model IM3* of the IM3-Control component, which is required to control the mapping of IM2 to IM3.

The third issue: whether there are structure and content within IM2-Control and whether IM3-Control is comprehensive enough to handle the transformations of IM1 to IM2 and IM2 to IM3. This issue directly correlates with the relevance of



Domain Knowledge Discovery Layer
(Includes Motivation and Strategy Layers of ArchiMate)
IM1-Control
Domain theory
Domain analyst
Domain knowledge elicitation cycle
Motivation and Strategy Elements
Capability
Resource
Management functional dependencies
Business management transactions
Domain causality description
IM1 – Real World Domain Knowledge Model
(RW is perceived as a self-managed system)
IM1 (r)
IM2-Control
(Business Analyst)
State of IM2 (y)
Instructions (u)
CIM Layer
(Business Layer of ArchiMate)
Internal Behavior Elements
Process
Management function
Elementary Management Cycle
Internal Structure elements
Business Object
Business Actor
Business Role
IM2 (r)
Business Service*
IM2 – Internal model of enterprise
(Enterprise is defined as a self-managed system)
(CIM - Computational independent model)
IM3-Control
(Architect / Designer)
State of IM3 (y)
PIM Layer
(Application Layer of ArchiMate)
Specifications (u)
Autonomic Component
Application Process
Application Function
Application Service
Application Component
IM3 – Enterprise software system architecture model
(ESS is specified as an autonomic system)
(PIM - Platform Independent Model)

Fig. No. 8. ***The internal modeling paradigm is illustrated using a modified MDA schema.***

domain knowledge discovery method and traceability (top-down and vice-versa) of the essential features of the domain between the layers. Furthermore, the internal model control system includes two feedback loops to ensure a self-management capability (see Fig. No. 1). An external feedback cycle is between the system (S) and IMC system, and the second (internal) feedback cycle is inside IMC between the Controller (C) and the Internal Model (IM).

Consequently, the cross-layer transformations (IM1 to IM2, IM2 to IM3, and IM3 to IM4) in Fig. No. 4 are under the control of the IM-Control components (IM1-Control, IM2-Control, and IM3-Control). The external feedback cycles are between the relevant IM-Control and IM of the lower level, i.e., between IM2-Control and IM2, and IM3-Control and IM3. The second (internal) feedback cycle is inside the IM-Control blocks. An example of the implementation of the internal model IM3* of the IM3-Control system is a framework of the autonomic computing component presented in Fig. No. 10.

An analysis of the cross-layer dependency between IM3 and IM4 (i.e. mapping PIM to PSM in Fig. No. 8) is out of the scope of this article.

The assumption three about the transferring of the essential features of the RW domain on the lower layers of modeling proven by the modified MDA schema in Fig. No. 8. Here an enterprise domain is perceived as a self-managed system in the context of second-order cybernetics (Glanville et. Al. 2002). Enterprise management activities are specified using *management functional dependency, management transaction (MT), and elementary management cycle* (EMC) concepts introduced in (Gudas, 2016), (Gudas, 2012a). The principle of the Internal Model Control (Fig. No. 1) is explored for cross-layer mapping control (Fig. No. 8). The cross-layer mapping is defined as the transformation of the complex structures, assuring the traceability of the causal dependencies (i.e., regularities fixed on the upper layer) between the layers, starting from IM1. The causality of RW domain is captured in IM1 and transferred and transformed for its intended purpose on the lower layers of the framework. For instance, the similarities of **the internal** architecture of MT and EMC frameworks (IM2) are reflected in the autonomic computing conceptual model (IM3) depicted in Fig. No. 10.

Examples in Fig. No. 7 and Fig. No. 8 highlight the qualitative differences of the two modeling paradigms:

- In the case of external modeling, RW domain perceived as the needs of the domain (stakeholder), which specified as an empirical concept map (IM1). However, in this way, RW domain knowledge becomes fragmented and relies on the experience of the business analyst, because the use of a set of the key concepts (e.g., *requirements*, *capabilities*) is not sufficient to capture RW causality (i.e., deep knowledge). The cross-layer relationships are based on the mapping concept to concept, but not on the transference of systems regularities by mapping structure to structure.
- In the case of internal modeling, RW domain is perceived as a complex system on the domain-related theoretical basis; in this way, a complex system captures the essential (deep) knowledge of the domain and specifies as IM1; then, it transmits through all layers due to the IMC-based cross-layer transformations.

By setting up the internal modeling paradigm in the modified MDA scheme for the enterprise domain (Fig. No. 8), it was established that:

- Fig. No. 6 presents the method of enterprise management knowledge discovery and provides a view of what is captured in IM1 – the essential content of management activities: enterprise goals, management information, data, data/knowledge transformations.
- The knowledge transfer between layers (a cross-layer mapping) is iterative, includes a feedback flow (y), and that corresponds with the principle of the internal model control system (Fig. No. 1).
- On a CIM layer, an internal model IM(2) is considered as a complex system (self-managed, goal-driven). The conceptualization of IM2 by using management transaction (MT) and EMC frameworks (Fig. No. 6) is relevant to capture the essential features of the domain.
- On a PIM layer, an internal model IM(3) is illustrated through the generalized architecture of the intelligent agent (Fig. No. 3) and the autonomic computing component (Fig. No. 10).

The intelligent agents and autonomic computing components are major technologies, used for the implementation of the internal modeling paradigm.

4.3. Capabilities of the Intelligent Agents

Autonomic computing components mostly are implemented using agent technologies (Kephart et al., 2003). Multiple autonomic managers of the software systems can learn, collaborate, and act independently with a degree of automatism.

Software agents allow delegation of tasks to the agents. We want to delegate integration and interoperability tasks to agents. For this purpose, the agent must understand domain environment and have an internal model of this environment. There are multiple types of agents: Simple reflex agents, model " based reflex agents, goal " based agents, utility based agents, and learning agents. Agents can also be reactive and proactive. There are few types of intelligent agents such as collaborative agents, interface agents, mobile agents, information/internet agents, reactive agents, hybrid agents, smart agents (Georgakarakou et al.). We are only focusing on the intelligent agents with the internal model (IM) of the environment as follows: Goal-based agents, Utility-based agents, learning agents. These agents are usually classified to be hybrid agents, smart agents, and believable agents.

The capabilities of the intelligent agents (Fig. No. 9) are classified as follows:

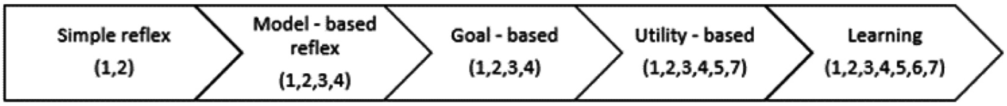


Fig. No. 9. ***The capabilities of the intelligent agent's types.***

1. Monitors states of data processing in applications;
2. Reacts to the specific state of data processing if necessary;
3. Understands data structure in each connected application (application environment);
4. Understands/perceives application processes (when and which data to use);
5. Uses utility functions to check structure changes proactively;

6. Creativity: determines and fixes problems in different applications;
7. Learning ability: has internal simulation/testing abilities, is able to optimize one's behavior;

The conceptual structure of the intelligent agents meets the generalized structure of the software component with the internal model (Fig. No. 3). All types of the intelligent agents include a domain model (environment model) as a set of rules needed to follow under certain conditions. The internal model of different intelligent agents captures the various knowledge items of the domain:

- Th IM of the model-based reflex agent includes a state of the world, a set of actions, a set of condition-action rules;
- The IM of the goal-based agent includes a state of the world, a set of actions, goals, decision-making element;
- The IM of the utility-based agent includes a state of the world, utility function, a set of actions, decision making element;
- The IM of the learning agent includes a state of the world, a learning sub-system, a performance element.

The capabilities (1–7) of the intelligent agents are corelated with the complexity of the internal model (IM) of the agent type. As a rule, the content of an IM of the intelligent agents is determined empirically; based on the experience of stakeholders (analysts, designers, and programmers), as a rule, it does not explore the fundamental theories of some particular domain type. Only the theoretical knowledge of a particular domain type is explored, and the resulting IM encapsulates causal dependencies and could be classified as a gray box or a white box.

4.4. Autonomic Computing Components

Autonomic computing systems are aimed to overcome growing software management complexity by introducing self-management capabilities (Kephart et al. 2003). The autonomic computing approach is an example of applying control theory concepts in software applications. It is already known that control theory-based approaches can be useful in a dynamic environment for the development of software to monitor and manage the behavior of system elements (Gaudin et al. 2011). Autonomic computing technologies exhibit four "self-management" characteristics (Kephart et al. 2003). First, self-configuration (able to configure its parameters) (Peukert et al. 2012; Feinerer 2007). Second, self-optimization (ability to reach optimal functioning). Third, self-healing (ability to restore work after disturbances). Finally, self-protection (ability to avoid disturbances/stay secure) (Heubscher et al. 2008; Parashar et al. 2005).

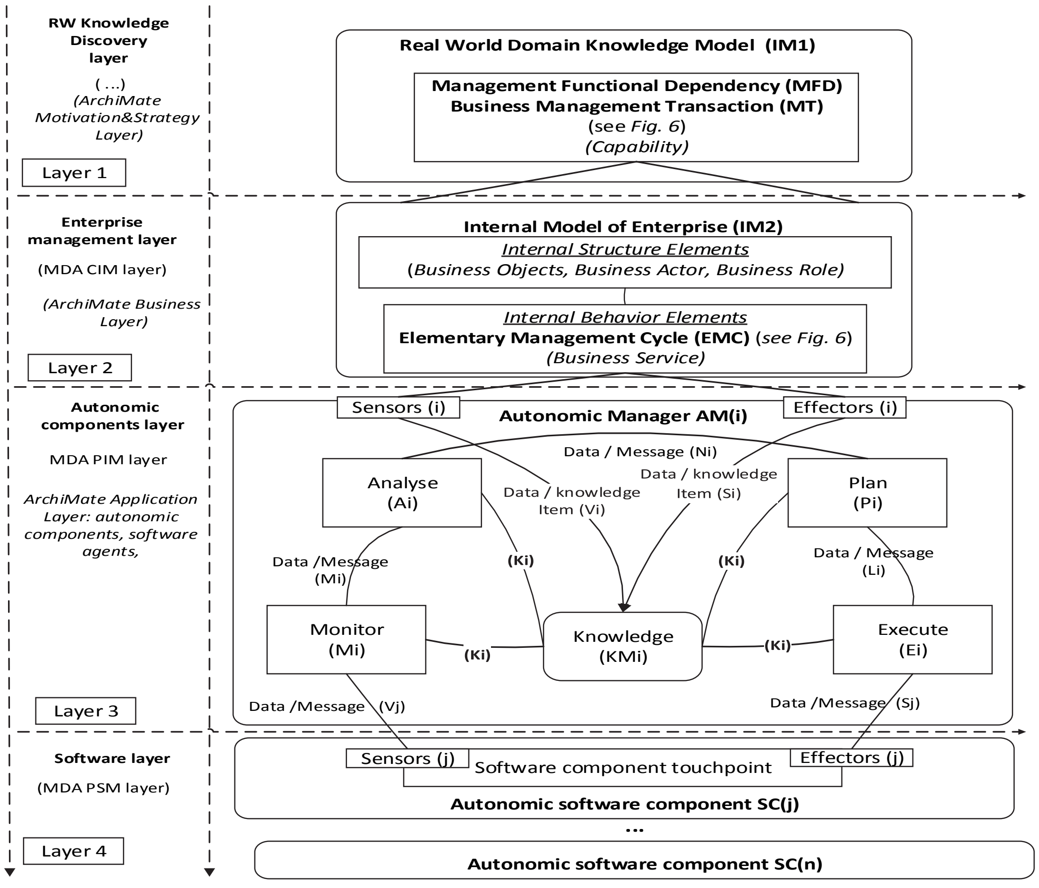
The elements of the autonomic computing component are as follows: the Monitor (M), Analyze (A), Plan (P), Execute (E) and Knowledge model. The Knowledge Model encapsulates knowledge of the situation and environment: rules, constraints, policies, and facts (Kephart et al. 2003). The content of the Knowledge model helps the elements M, A, P and E to recognize states and eventually respond to changes.

Fig. No. 10 depicts a version of the autonomic manager specialized for enterprise management; it is developed using the internal modeling paradigm and the enterprise management frameworks discussed above. This autonomic manager includes the knowledge model KM, which is the result of cross-layer transitions starting from the internal model of enterprise domain (IM1)

(Fig. No. 6). So, by using our approach, we obtain the following results: the autonomic manager on the application layer (Fig. No. 10) and the enterprise management frameworks (MT and EMC, in Fig. No. 6) on the business layer are similar conceptual structures of the self-managed systems, but on the different layers of modeling.

The ArchiMate model exchange format (MEFF) was used for transforming IM (business layer) to KM (application layer) (The Open Group, ArchiMate® Model Exchange File Format 2015). Using MEFF, the business model reduces to a set of rules understandable for the autonomic manager. Autonomic managers (AM) may form a hierarchical structure: a lower level AM(i) is controlled by upper-level AM(i-1) and so on.

The architecture of the enterprise management system with the autonomic manager (AM) in Fig. No. 10 reveals that conceptualizations in Fig. Nos. 6 and 10 are in line with each other, and this is a validation of the third assumption. The conceptual structure of the enterprise management framework (elements of EMC) matches the components of the autonomic manager contains eight matching components. Interpretation (IN) matches the component Moni-



RW Knowledge Discovery layer
(...)
(ArchiMate Motivation&Strategy Layer)
Layer 1
Real World Domain Knowledge Model (IM1)
Management Functional Dependency (MFD)
Business Management Transaction (MT)
(see Fig. 6)
(Capability)
Enterprise management layer
(MDA CIM layer)
(ArchiMate Business Layer)
Layer 2
Internal Model of Enterprise (IM2)
Internal Structure Elements
(Business Objects, Business Actor, Business Role)
Internal Behavior Elements
Elementary Management Cycle (EMC) (see Fig. 6)
(Business Service)
Autonomic components layer
MDA PIM layer
ArchiMate Application Layer: autonomic components, software agents,
Layer 3
Sensors (i)
Autonomic Manager AM(i)
Effectors (i)
Data / Message (Ni)
Data / knowledge Item (Si)
Data / knowledge Item (Vi)
Analyse (Ai)
Plan (Pi)
Data /Message (Mi)
(Ki)
Data / Message (Li)
Monitor (Mi)
Knowledge (KMi)
Execute (Ei)
Data /Message (Vj)
Data /Message (Sj)
Software layer
(MDA PSM layer)
Layer 4
Sensors (j)
Software component touchpoint
Effectors (j)
Autonomic software component SC(j)
...
Autonomic software component SC(n)

Fig. No. 10. ***Enterprise Architecture with autonomic manager based on the internal modeling paradigm.***

tor (M), Data Processing (DP) matches the component Analyze (A), Decision Making (DM) matches the component Plan (P), the Realization of decision (RE) corresponds to the component Execute (E), respectively. The autonomic manager AMi on the layer three focus on the control of the software components SC(j) on layer 4 (i.e., it could be the enterprise applications (e.g. CRM, E-Commerce). A control loop for control of software component SC(j) comprises Monitor (Mi), Analyze (Ai), Plan (Pi), and Execute (Ei) components and the data/ message flows Sj, Li, Ni, Mi, and Vi. The feedback (information flows Ki) between knowledge component (KMi) and other elements of the autonomic manager (AM(i)) ensures a self-management capabilities (Fig. No. 10). The feedback information flows Ki, from components M, A, P, E to KMi, contain information of the previous task's execution on event logs (start, stop and error logs).

Enterprise management applications controlled by the autonomic manager over web services or direct data connections to the database. Sensors in Fig. No. 10 are web services or direct data access points that allow data extraction (using Get operations). Effectors are web services or direct data access points that allow data input (using Set operations) (Fig. No. 10). An autonomic manager retrieves data through data flow Vi (Get operation) and pushes data through data flow Si. The component Monitor (Mi) follows the content of domain knowledge captured in the Knowledge component (KMi). The business layer data and knowledge items obtained through data/knowledge flows (Vi, Si). The result of monitoring is a message Ji passed by component Monitor (Mi) to component Analyse (Ai). The autonomic manager-based architecture of the interoperable enterprise applications presented in the next section.

5. Approach to Application Interoperability using the Internal Model

5.1. The Problem of Applications Interoperability

In this section, the issue of interoperability of applications is discussed in more detail. In the real world enterprise scenario, a specific software system (a set of applications and databases, mainframes, workstations, data) supports some business processes (e.g., customer registering, manufacturing, selling, shipping). The application requires a feedback loop, determined for the mutual interaction scenario of some business process (Fig. No. 11). An example of the business activity scenario: a sort of a manufacturing plant uses devices with sensors to observe manufacturing processes, product testing, and packing processes. The software system of each manufacturing facility has the multiple interfaces to receive data from the sensors. Our approach is different from other interoperability methods, because, in this paper, we research a dimension that slices through three distinct interoperability levels (Technical, Semantic/Data and Organization); these levels are clearly defined in the European Interoperability Framework (EIF) and mentioned in multiple other articles (EIF 2004; F.B. Vernadat 2007).

In a dynamic enterprise environment, applications might be changed and adapted following the business requirements changes, so the business process model should be modified as well. Consider a case of business changes, where a new generation device (collecting robot or some other new device) is installed (Fig. No. 11). A new application (depicted in Fig. No. 11 as a pair (F(new), P(new)) has different interfaces or slightly different data formats that have not been registered in the plant database before. The challenge is that the new device can cause changes in the business process flow, and efficiency problems can appear. For instance, additional work will be required to integrate with the legacy software if a new device is not able to adapt itself correctly. The manufacturing staff (users) and programmers have to work together to modify interfaces of the existing software systems due to that new installation, ensuring full interoperability among the new and old software systems.

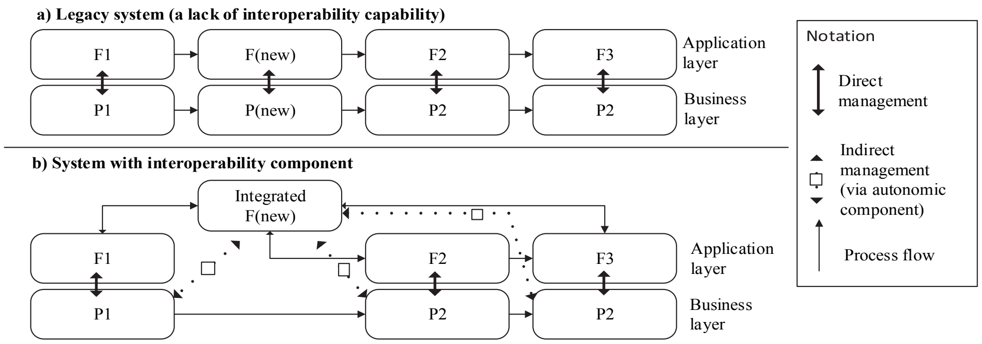


Figure No. 11. ***Interoperability in the enterprise system.***

If an existing (legacy) software system is not flexible enough to handle with changes (Fig. No. 11a – a lack of interoperability capability), additional efforts are required for the software system integration when the changes occurred. The legacy applications would not be able to communicate without introducing a new device to all the existing software systems of the enterprise. A typical case of the required efforts to ensure the integrity of the software system of any organization is the work put in to restore business and software integration. Restoring integration is an iterative process as follows: to find the issue – to understand the issue – to fix the issue.

The integration of the new installations and the existing software systems is the issue. Fig. No. 12 presents the architecture of the autonomic integration system. It is developed using as a background the modified MDA (Fig. No. 8), the knowledge components of enterprise domain (Fig. No. 6) and autonomous computing components (Kephart et al. 2003).

Fig. No. 11 presents the system with the interoperability component. Looking to the enterprise system from the perspective of the internal modeling based MDD (the modified MDA scheme in Fig. No. 8), we found out that two-way communication between the business layer and application layer is required. The cross-layer feedback loop ensures system integrity and is determined using knowledge models (IM2 and IM3 in Fig. No. 8).

The new application *Integrated F(new)* is an autonomic component (application layer in Fig. No. 11b), which is able for self-integration with existing (legacy) applications due to the internal knowledge (captured in KMi, see Fig. No. 10). In this case, due to such functionality of the self-integration of the *Integrated F(new),* there is no need for changes in the existing business processes (Business layer in Fig. No. 11b).

5.2. State of the Art in Application Interoperability Solutions

There are a lot of different types of the enterprise application interactions in the dynamic environment (e.g. Customer entry to the application, order placement). To maintain the application interoperability is complicated if data structures or the web service composition are not available.

Method to find the best solution for designing the interoperability of enterprise applications is described in (Galasso et al. 2016). The important point is that based on accurate and relevant business process model the measurement of interoperability performance. Presented in (Galasso et al., 2016) methods are focused on the evaluation of the complexity of interoperability projects and choices of the best interoperability solution based on the business process modeling.

In dealing with the applications interoperability problem, Papazoglou et al. (2008) declare a need for service-oriented computing, known as SOC. However, they do not mention problems of application communication difficulties (between web services or schema alignment) (McCann et al. 2005), record linkage, data fusion (Dong et al. 2013), application communication orchestration or choreography.

Dervice-oriented architecture (SOA) is used to define communication of the web services (Krafzig et al. 2005; Michlmayr et al. 2007). However, the web service itself does not communicate with other systems without medium application. Middleware integration application defines how and when data migrate and perform migration actions from one web service to another. B. Benatalah et al. (Benatallah et al. 2005) analyzed the requirements of the special adapters to web services to integrate enterprise applications. However, the authors do not mention how to solve interoperability issues in a dynamic enterprise environment when the application structure changes.

Neither Lankhorst (Lankhorst 2013) nor Open Group (Georgakarakou et al.) provided a detailed description of the application collaboration. In the Open Group documentation, it seems that a collaboration element can only be collaborating with the components of the same application but not with the elements of different applications.

In a common case, the applications do not have direct access to use the inner components of other applications and thus are not able to ensure interoperability on the component level (without external impact). When examining the SOA API interface specifications, we can determine the interface data structures and their types, but the data attributes matching can not be identified. Furthermore, the SOA API interface specifications not determine the sequence of actions (which should define the flow of integration with each application). However, business process model helps to discover such sequence. Since it is impossible to obtain an internal data structure of other application, the alternative is to use the detailed (deep knowledge) captured by the domain model. Only using a deep knowledge would allow determining the integration actions and the sequence of actions.

The modified MDA (Fig. No. 8) based approach to the autonomic integration was developed using the knowledge components of enterprise domain (Fig. No. 6) and autonomous computing components (Kephart et al., 2003). This applications interoperability solution based on the deep domain knowledge model. Fig. No. 12 depicts the architecture of the interoperable enterprise applications. Considering the internal modeling perspective applied at the each layer of modified MDA (Fig. No. 8), the internal models (IM2, IM3, IM4) preserve the mappings of the essential dependencies of the particular RW domain, which are captured at the top layer (knowledge discovery layer) and fixed as IM1.

5.3. The Architecture of the Interoperable Enterprise Applications

The architecture of the interoperable enterprise applications presented in Fig. No. 12. The key element of the solution is a middleware called the autonomic interoperability application. Autonomic interoperability application acts as a medium between multiple legacy applications Application 1, …, Application n (Fig. No. 12).

The enterprise model IM2 (MDA CIM, or ArchiMate business layer in Fig. No. 8) describes enterprise management activities and enterprise environment. The most important part of the enterprise model IM2 is the business activity sequence (workflow) of the management transaction, specified in detail by an elementary management cycle (Fig. No. 8). IM2 is content of the knowledge element KM2 of the autonomic interoperability application. Business process flow rules might also be derived from WSDL file of web service (Valatavicius et al. 2014). Also, the data might duplicate in the different applications of a single enterprise, the data structures, and fields naming can be heterogeneous (that require changing format) (Bernstein et al. 2011)).

What is more complicated, the application data management process can be different, and this is the reason why the business process model is so important for retracing the sequence of data management events. Therefore, business modeling language should describe what data and in what order transferred between applica-

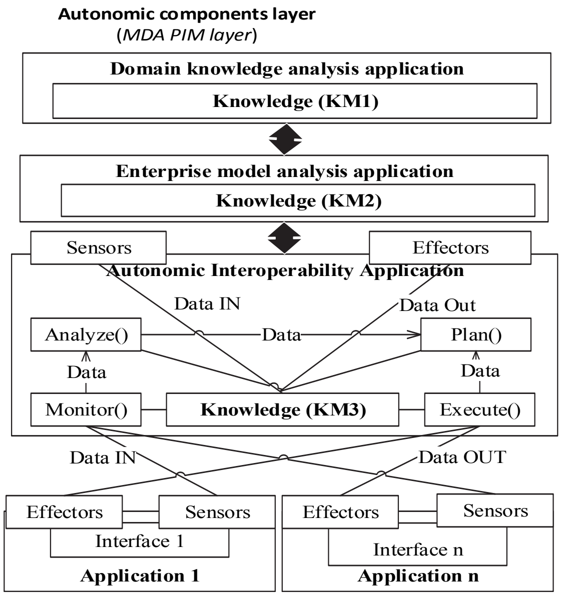


Fig. No. 12. ***The architecture of the interoperable enterprise applications.***

tions. The collaboration element should use application interfaces, not application components. The modeling languages (e.g., UML, BPMN, ArchiMate) discussed in the previous chapter have limited capabilities for the specification of the application collaboration element.

The relation between the business process model and application management process is explained by creating an internal model of relationships between them. The internal model should remain as a ruleset in the knowledge elements (KMi) of each autonomic component. The architecture of the interoperable enterprise applications is depicted in Fig. No. 12, and it includes the autonomic interoperability application (AIA). AIA monitors other application interfaces (two or more) for data records changes using web service interfaces. AIA transfers the modified data copies to other applications. The required business process flow is identified by AIA using the knowledge elements (KMi).

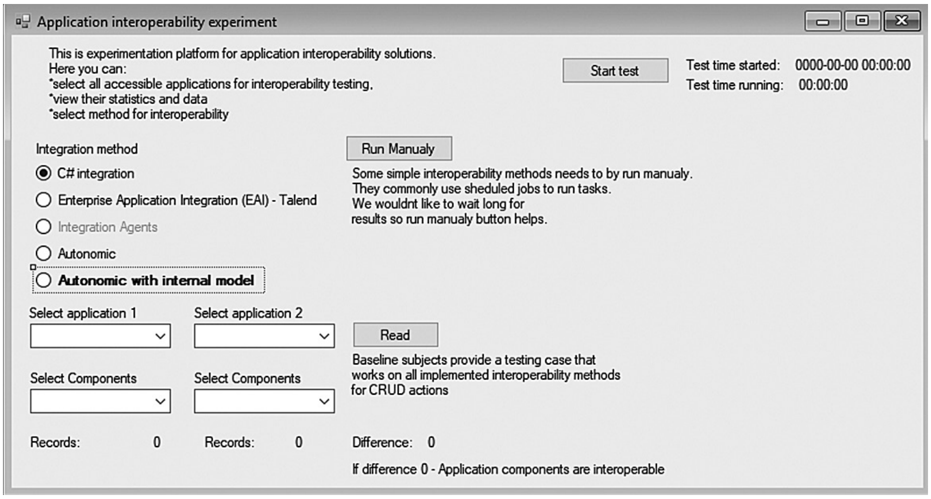
The autonomic computing element stems from IBM autonomic computing methodology (Jackob et al. 2004). The knowledge element must contain basic rules and policies to have self-management capabilities. To record the state of the integrated applications in a dynamic business environment, the autonomic component has to monitor sensors placed in managed applications.

The presented in Fig. No. 12 autonomic application can collaborate with other applications related with the same domain or can require an additional input (knowledge).

6. The Prototype of the Software Interoperability Validation

The prototype version for testing of enterprise application integration solutions is under development (the screenshot in Fig. No. 13). Currently, the prototype calculates the number of the records in applications for each integration method. If the difference is zero (=0) the specified application component is interoperable, if the difference is not zero (> 0), it is not interoperable.

However, this prototype does not cover some issues of schema matching. For the experimental verification of proposed



This is experimentation platform for application interoperability solutions.
Here you can:
*select all accessible applications for interoperability testing,
*view their statistics and data
*select method for interoperability

Test time started: 0000-00-00 00:00:00
Test time running: 00:00:00

Integration method
[✓ C# integration]
Enterprise Application Integration (EAI) - Talend
Integration Agents
Autonomic
Autonomic with internal model

Some simple interoperability methods needs to by run manualy.
They commonly use sheduled jobs to run tasks.
We wouldnt like to wait long for
results so run manualy button helps.

Select application 1 | Select application 2

Baseline subjects provide a testing case that
works on all implemented interoperability methods
for CRUD actions

Select Components | Select Components

Records: 0 | Records: 0 | Difference: 0

If difference 0 - Application components are interoperable

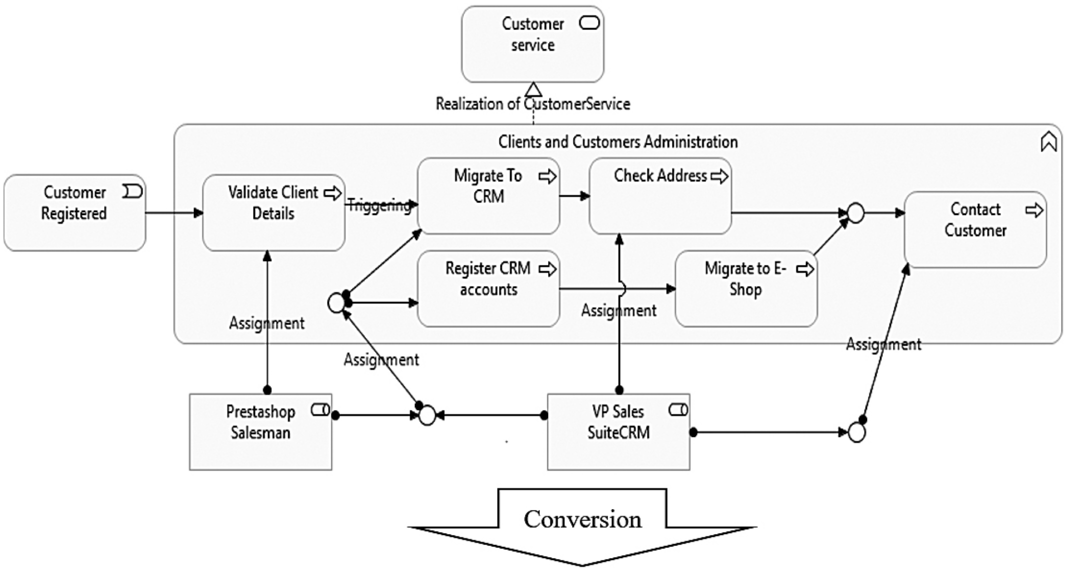
Fig. No. 13. ***Screenshot of the prototype for software interoperability validation.***

solution of the autonomic interoperability component with the internal model (Fig. No. 12), we prepared a simple ArchiMate business layer model that covers processes of the fictional organization (Fig. No. 14). This model covers only a very tiny part of the processes (i.e., only the registration of clients, customers or suppliers). The business model depicted in Fig. No. 14 is extracted using the model exchange format of the Dublin Core schema version 1.1. Fig. No. 14 presents specifications extracted from this document as part of knowledge content required for the autonomic computing element KM3. KM3 stores the real-world knowledge (Fig. No. 14) in a Model Exchange File Format (MEFF).

On an experimental basis, we can say that:

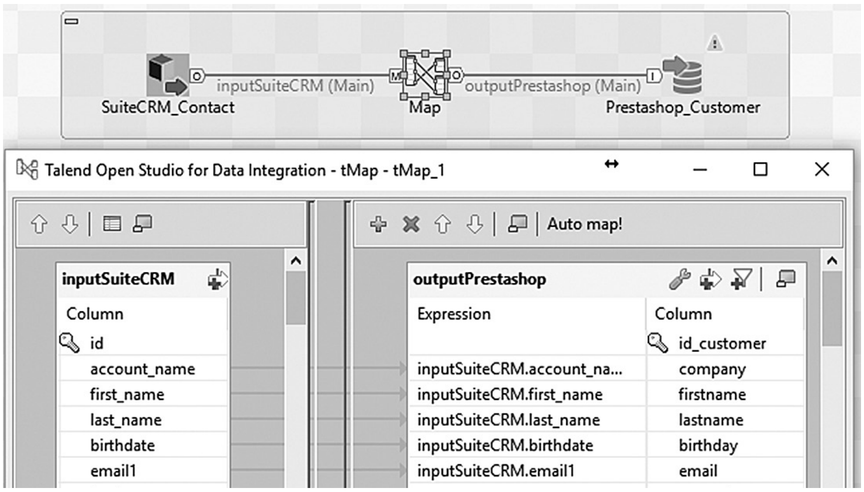
- By using native code integration solutions (i.e., c# interoperability solution), the complex logic required for interoperability application middleware can be achieved, but all manageability efforts belong to the programmer. Integration specialist or interoperability administrator should manually implement every new adaptation to the environment.
- The development of the interoperability solution is easier with enterprise application integration (EAI) due to a graphical designer. Here, the scheme of all application components is visualized and can be mapped easily. The manageability level is higher than using native code.

In Fig. No. 15a, we present the choreography of one-way interoperability of two



```
- <element identifier="id-41bda1c1-9f41-406e-8a6a-c3ab9f5754ff" xsi:type="BusinessFunction">
      <name xml:lang="en">Clients and Customers Administration</name>
  </element>
- <element identifier="id-b3466b18-dfc9-40f5-83d6-7146531869fa" xsi:type="BusinessProcess">
      <name xml:lang="en">Validate Client Details</name>
  </element>
- <element identifier="id-476f670e-6f94-4bfb-9fb4-032e8328061b" xsi:type="BusinessProcess">
      <name xml:lang="en">Register CRM accounts</name>
```

Fig. No. ***14. Business architecture layer covering registration of clients and its conversion to the MEFF format.***



Talend Open Studio for Data Integration - tMap - tMap_1

inputSuiteCRM	outputPrestashop	
Column	Expression	Column
id		id_customer
account_name	inputSuiteCRM.account_na...	company
first_name	inputSuiteCRM.first_name	firstname
last_name	inputSuiteCRM.last_name	lastname
birthdate	inputSuiteCRM.birthdate	birthday
email1	inputSuiteCRM.email1	email

Fig. No. 15. ***Choreography (a) and schema matching (b) in "Talend Open Studio for Data Integration" (EAI).***

different software components: the application's "SuiteCRM" component "Contacts" is integrated with the "Prestashop" component "Customer." In Fig. No. 16b, there is a specification of the Map element that describes the mapping of attributes (fields) of two different components in the various applications. The more fields corresponding to one of the other, the better the chances that the components are interoperable. In other words, these components are representations of the same entity of the real world. Choreography (Fig. No. 15a) is fully dependent on the business architecture (Fig. No. 14) and its elements "Migrate to CRM," "Migrate to E-Shop" dictates the execution order of components described in Fig. No. 14.

By the links between the component fields depicted in Fig. No. 15b, it can be concluded that the "Suite CRM" component "Contacts" is semantically interoperable with the "Prestashop" component "Customers." In our prototype, validation indicates when the difference of record count is zero (=0). For instance, with a new record in Contacts (CRM) created a new record for the same entity should appear in Customers (E-Shop), and the difference of record count is zero (=0) showing the interoperability succeeded.

Our research is still in progress, and we need to continue working on the autonomic interoperability component usage in the dynamic business environment.

Conclusions

The scientific contribution of our approach is a new viewpoint toward the interoperability of applications. Our research suggests that to achieve higher levels of autonomy, every smart system should encapsulate a deep knowledge of a target domain. By integrating the internal modeling paradigm with MDA approach and using this method to the development of interoperability solutions, we seek to create more autonomous software systems in the enterprise environment. The review of the modeling methodologies reveals the relationships between the business domain modeling paradigms, enterprise architecture modeling, software architecture modeling.

Our research shows that autonomic application interoperability can be achieved using IBM's autonomic computing approach together with a deep knowledge of the real world domain (i.e., the internal model), but the challenge is in the understanding how these models can be integrated together. Moreover, our research reveals the main perceived causality of the target domain at the enterprise architecture modeling and current implementation of applications in business. From our research, it is clear that the software engineering target domain is an enterprise – a complex organizational system. Other findings state that functional management dependencies of the management activities are the essential knowledge in the business domain required for business software engineering. In practice, no model of enterprise architecture and business models are used before designing and developing interoperability between multiple applications. Our solution suggests that the models be created using a modified MDA approach; enterprise architecture and business process models can be used to reach higher levels of autonomy of interoperable application solutions. The constructed theoretical background includes internal modeling paradigm definitions from second order cybernetics and autonomic computing approach, and it allows model autonomous integration as well as interoperability solutions. The relevance of the domain knowledge discovery method is currently the main roadblock to continuing our research and by itself is a fundamental issue of the domain modeling, which determines the relevance (validity) of IM against the RW domain. A theoretical background (domain theory) requires for recognizing the essential features of the domain type.

An analysis of the role of the internal model (IM) in control systems allows for concluding that the adaptation of the internal model (IM) in the context of software systems development is a relevant topic for enhancing intelligent technologies. The discrepancy of domain complexity and modeling capabilities causes problems of the enterprise applications development, integration, and adjustment to the environment changes. The reason for the deficiency is the modeling methodology, because enterprise domain is modeling languages is based on the external modeling paradigm. Such models are not focused on the modeling of the business dynamic (i.e., not focused on the domain causal dependencies), and therefore currently are inadequate to support the development of the intelligent enterprise software (e.g., autonomous applications). The prerogative is using the internal modeling paradigm.

Our paper contributes to the theory of application interoperability by proposing an interdimension approach of multiple integration levels (organizational, data/semantical, technical) mentioned in the European Interoperability Framework (EIF).

The internal modeling paradigm consolidation with the model, which is the driven architecture approach (OMG MDA), is described and illustrated. The peculiarity of the modified MDA is a focus on the cross-layer transferring of domain causality. The internal modeling concentrate on the discovering of deep knowledge of the problem domain, i.e., the internal modeling is aimed to reveal causal dependencies of the problem domain. The adapting of the internal modeling approach for enterprise domain modeling and intelligent software system development looks promising. The proposed modified MDA framework is based on the three assumptions as follows. First, the knowledge-based enterprise software development methods should be focused on the modeling of the causal dependencies of the domain. The second assumption of the software system development definition as the cross-layer mapping (e.g., motivation, business, application and technology layers) of the internal models confirmed by the layered structure of the enterprise architecture frameworks. The third assumption for the transference of essential features of the real world domain across the layers is a fundamental condition; it is shown by the similarity of the knowledge model CIM layer (MDA) and PIM level knowledge models. The capabilities of the intelligent software systems (applications) strongly depend on the real world domain causality discovering on the top layer and the cross-layer transferring of the identified causal dependencies. The cross-layer relationships in OMG MDA is a mild statement; it is characterized as the mapping of models (CIM to PIM, PIM to PSM). The cross-layer transferring of the deep knowledge in the context of the internal modeling paradigm (as well as the good regulator theorem) requires stricter definition.

In the case of external modeling, a real world domain is perceived in terms of certain stakeholder needs, which are revealed and specified as a concept map (IM1). In this way, real world domain knowledge is fragmented, only a few key concepts (e.g., the requirements and capabilities) are the background for the next stage of development. We discovered that the contents of all other layers depend on the main concepts of the upper layer, i.e., that they depend more on the experience of an analyst and the selected modeling method. So, by considering the internal modeling perspective, each layer of the modified MDA (Fig. No. 8) contains the transformed necessary dependencies of the domain (IM2, IM3 and IM4, respectively), which are captured and fixed as IM1 at the top layer (knowledge discovery layer).

We were able to find similarities betweem the internal model of enterprise domain (IM2), enterprise architecture model (IM3), and the autonomic computing component architecture. The similarities are namely the general internal structure (internal models) of these different types of systems; in particular, the similarities of the internal transactions (feedback loops), including the information and knowledge, flows in the feedback loops.

This internal modeling paradigm is consolidated with the model that is driven by the software development approach and is illustrated by a case study of the interoperability problems, using the autonomic computing components approach. The knowledge element of the autonomic component contains a complex model of the dynamic environment and controls the behavior of integration processes. This autonomic interoperability component is focused on evaluating the state of the other applications and ensure the integration of applications in a dynamic business environment.

The architecture of the interoperable enterprise applications with the autonomic integration component is presented and demonstrated by the prototype. However, further work is needed to make the comparison to existing interoperability solutions. The presented approach is different from other interoperability methods, because in this paper, we research a dimension that slices through three distinct interoperability levels (Technical, Semantic/Data, and OrganiZation). The assumption is that nothing can have a properly designed interoperability of enterprise applications if it has no knowledge of domain causality, which should be transferred across the modeling layers from the business process modeling to these enterprise applications development. In most rival articles on interoperability, there is a lack of analysis of the mutual relations of application and business processes. Therefore, this approach is aimed to get more insights into the autonomic interoperability subject, which would be based on the deep knowledge of the domain.

The experimental verification of the proposed method was made for an E-Shop environment using three software systems: Webshops (Prestashop and Oscommerce) and CRM (SuiteCRM). The ongoing experiment confirms that application integration and interoperability solutions are not an easy task, even in a static environment. There is still a lot of work to be done to gather evidence that autonomic interoperability application with the internal enterprise domain model is a reliable solution. With the initial prototype created for the validation of interoperability of applications, we observed that deep knowledge (internal model) is essential for effective interoperability.

REFERENCES

ABDELZAHER, Tarek, et al. (2008). Introduction to control theory and its application to computing systems. In: Performance Modeling and Engineering. Springer US, p. 185–215.

ArchiMate® 3.0 Specification [interactive], The Open Group, 2016. Document Number: C162 [reviewed 2017 y. June 15 d.]. Internet access: <http://pubs.opengroup.org/architecture/archimate3-doc/>. ISBN: 1-937218-74-4.

ASHBY, W. Ross (2017). *An introduction to cybernetics*. S. l.: London Chapman & Hall Ltd, 1956 [reviewed 2017 y. June 9 d.]. Internet access: <http://dspace.utalca.cl/bitstream/1950/6344/2/IntroCyb.pdf>.

BARTON, Rick (2013). *Talend Open Studio Cookbook*. Packt Publishing Ltd. ISBN:1782167277.

BENATALLAH, Boualem et al. (2005). *Developing adapters for web services integration*. In International Conference on Advanced Information Systems Engineering. Springer Berlin Heidelberg, p. 415–429.

BERNSTEIN, Philip A., et al. (2011). Generic schema matching, ten years later. In: *Proceedings of the VLDB Endowment,* 4(11) p. 695–701.

BRACHE, Alan P. (2002). *How organizations work: Taking a holistic approach to enterprise health*. John Wiley & Sons.

CONANT, Roger C.; ASHBY, W. (1970). Ross. *Every good regulator of a system must be a model of that system*. In: International journal of systems science, 1 (2), p. 89–97.

CZARNECKI, Krzysztof; HELSEN Simon (2003). Classification of model transformation approaches. In: *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 45(3).

DIETZ, Jan LG. (2006). The deep structure of business processes. *Communications of the ACM*, 49(5), p. 58–64.

DONG, Xin Luna; NAUMANN Felix (2009). Data fusion: resolving data conflicts for integration. In: *Proceedings of the VLDB Endowment*, 2(2), p. 1654–1655.

DONG, Xin Luna; SRIVASTAVA, Divesh (2013). *Big data integration*. In: Data Engineering (ICDE), IEEE 29th International Conference, p. 1245–1248.

European interoperability framework for pan-european egovernment services. European Communities, 2004 [reviewed 2017 y. June 3 d.]. Internet access: <http://ec.europa.eu/idabc/servlets/Docd552.pdf>. ISBN 92-894-8389-X.

EL-HALWAGI, Mahmoud M. (2016). *Process integration*. Academic Press, 7. ISBN 0-12-370532-0.

FAYOL, Henri (2016). *General and industrial management*. Ravenio Books.

FEINERER, Ingo (2007). A formal treatment of UML class diagrams as an efficient method for configuration management.

FOWLER, Martin. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

FRANCIS, Bruce A.; WONHAM, W. Murray (1976). The internal model principle of control theory. *Automatica*, 12(5), p. 457–465.

GALASSO, François, et al. (2016). A method to select a successful interoperability solution through a simulation approach. *Journal of Intelligent Manufacturing*, 27(1), p. 217–229.

GAUDIN, Benoit, et al. (2011). Nixon A Control Theory-Based Approach for Self-Healing of Unhandled Runtime Exceptions. In: *Proceeding of the 8th ACM international conference on Autonomic computing*, ACM, p. 217–220.

GEORGAKARAKOU, Chrysanthi E.; ECONOMIDES, Anastasios A. (2003). Software Agent Technology: an Overview Application to Virtual Enterprises. In: *Agent and Web Service Technologies in Virtual Enterprises*, N. Protogeros (ed.). Idea Group Publ.

GEORGAKOPOULOS, Diimitrios; HORNICK, Mark; SHETH, Amit (1995). An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2), p. 119–153.

GLANVILLE, Ranulph (2002). Second order cybernetics. In: *Systems Science and Cybernetics*, p. 59–85.

GROSSMANN, Georg; SCHREFL Michael; STUMPTNER, Markus (2007). *Exploiting semantics of inter-process dependencies to instantiate predefined integration patterns*. Australian Computer Society, p. 155–156.

GUDAS, Saulius (2016). Information Systems Engineering and Knowledge-Based Enterprise Modelling: Towards Foundations of Theory. In*: Springer Proceedings in Business and Economics*, p. 481–497. ISBN 978-3-319-33865-1.

GUDAS, Saulius; LOPATA, Audrius (2016). Towards internal modelling of the information systems application domain. *Informatica,* 27(1), p. 1–29. ISSN 0868-4952.

GUDAS, Saulius; LOPATA, Audrius (2015). Meta-model based development of use case model for a business function. *Information Technology and Control*, 36 (3).

GUDAS, Saulius (2012). *Foundations of the information systems' engineering theory*. Vilnius University Press, p. 384.

GUDAS, Saulius (2012). Knowledge-Based Enterprise Framework: A Management Control View. In: *New Research on Knowledge Management Models and Methods*. InTech, 2012.

HALEVY, Alon; RAJARAMAN, Anand; ORDILLE, Joann (2006). Data integration: the teenage years. In: *Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment*, p. 9–16.

HEYLIGHEN, Francis; JOSLYN, Cliff (2001). Cybernetics, and Second-Order Cybernetics. In: *Encyclopedia of Physical science & Technology*, 4, p. 155–170.

HOHPE, Gregor; WOOLF, Bobby (2002). *Enterprise integration patterns*. In: *9th Conference on Pattern Language of Programs*, p. 1–9.

HUEBSCHER, Markus C.; MCCANN, Julie A. (2008). A survey of autonomic computing-degrees, models, and applications. *ACM Computing Surveys (CSUR),* 40(3), p. 7.

JACOB, Bart, et al. (2004). A practical guide to the IBM autonomic computing toolkit. In: *IBM Redbooks,* 4, p. 10.

KARDOŠ, Martin; DROZDOVÁ, Matilda (2010). Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). *Journal of Information and Organizational Sciences*, 34(1), p. 89–99.

KEPHART, Jeffrey O.; CHESS, David M. (2003). The vision of autonomic computing. In: *Computer,* 36(1), p. 41–50.

KUMAR, Shrawan (2012). *Kac-Moody groups, their flag varieties, and representation theory*. Springer Science & Business Media.

KUTSCHE, Ralf-Detlef; MILANOVIC Nikola, eds. (2008). Model-Based Software and Data Integration: First International Workshop. In: *Proceedings*, vol. 8. Springer Science & Business Media. MBSDI.

KRAFZIG, Dirk; BANKE Karl; SLAMA, Dirk (2005).. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional.

KROGSTIE, John (2005). EEML2005: extended enterprise modeling language. Norvegian University of Science and Technology.

LABROU, Y. Peng1, et al. (1998). *A multi-agent system for enterprise integration.*

LANKHORST, Marc (2013). Communication of Enterprise Architectures. In: *Enterprise Architecture at Work,* p. 61–74.

LI, Li; WU, Baolin; YANG, Yun (2005). Agent-based ontology integration for ontology-based applications. In: *Proceedings of the 2005 Australasian Ontology Workshop,* vol. 58. Australian Computer Society, Inc., p. 53–59.

MAREELS, Iven; POLDERMAN, Jan Willem (2012). *Adaptive systems: an introduction*. Springer Science & Business Media. ISBN 978-1-4612-6414-9.

MCCANN, Robert, et al. (2005). Mapping maintenance for data integration systems. In: *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, p. 1018–1029.

MEDINA-MORA, Raul, et al. (1992). The action workflow approach to workflow management technology. In: *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. ACM.

MICHLMAYR, Anton, et al. (2007). Towards recovering the broken SOA triangle: a software engineering perspective. In: *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*. ACM, p. 22–28.

MOEN, Ronald; NORMAN Clifford (2006). *Evolution of the PDCA cycle*.

OSIS, Janis (2004). Software development with topological model in the framework of MDA. *CAiSE Workshops*, 1, p. 211–220.

OVEREINDER, Benno J.; VERKAIK, P. D.; BRAZIER, Frances MT (2008). Web service access management for integration with agent systems. In: *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, p. 1854–1860.

PAPAZOGLOU, Michael P., et al. (2008). *Service-oriented computing: a research roadmap*. *International Journal of Cooperative Information Systems,*17(2), p. 223–255.

PARASHAR, Manish; HARIRI Salim (2005). Autonomic computing: An overview. In: *Unconventional Programming Paradigms*, p. 97–97.

PAVLIN, Gregor; KAMERMANS, Michiel; SCAFES, Mihnea (2010). Dynamic process integration framework: Toward efficient information processing in complex distributed systems. *Informatica*, 34(4).

PEUKERT, Eric; EBERIUS Julian; RAHM Erhard (2012). A self-configuring schema matching system. In: *IEEE 28th International Conference on Data Engineering*. IEEE, p. 306–317.

PORTER, Michael E.; MILLAR Victor E. (1985). *How information gives you a competitive advantage*.

RAHM, Erhard; BERNSTEIN, Philip A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, p. 334–350.

SENDALL, Shane; KOZACZYNSKI Wojtek (2003). Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5), p. 42–45.

SHVAIKO, Pavel; EUZENAT Jérôme (2013). *Ontology matching: state of the art and future challenges*. In: IEEE Transactions on knowledge and data engineering, 25(1), p. 158–176.

SILVERSTON, Len; INMON, William H.; GRAZIANO Kent (1997). *The data model resource book: a library of logical data models and data warehouse designs*. John Wiley & Sons, Inc. ISBN:0471153672.

TIN, Chung; POON, Chi-Sang (2005). Internal models in sensorimotor integration: perspectives from adaptive control theory. *Journal of Neural Engineering*, 2(3), S147.

TROTTA, Gian (2003). *Dancing Around EAI'Bear Traps'*. Business Process Management (BPM) Best Practices.

VALATAVICIUS, Andrius; DILIJONAS, Darius (2014). Dynamic B2B process integration. In: *Proceedings of "Informacinės Technologijos"*, p. 34–39.

VALATVICIUS, Andrius; GUDAS, Saulius (2015). *Enterprise Software System Integration Using Autonomic Computing*. CEUR-WS. org, 1420, p. 156–163.

VAN DEN BOSCH, Marcel APM, et al. (2010). A selection-method for Enterprise Application Integration solutions. In: *International Conference on Business Informatics Research*, p. 176–187.

VERNADAT, François (2002). UEML: towards a unified enterprise modelling language. *International Journal of Production Research*, 40(17), p. 4309–4321.

VERNADAT, François B. (2007). Interoperable enterprise systems: Principles, concepts, and methods. In: *Annual Reviews in Control*, 31(1), p. 137–145.

WINOGRAD, Terry; FLORES Fernando (1986). *Understanding computers and cognition: A new foundation for design*. Intellect Books.

WHITE, Stephen A., et al. (2011). *BPMN 2.0 handbook second edition: methods, concepts, case studies and standards in business process modeling notation*. Future strategies, Inc.

WINTER, Kirsten; SANTEN Thomas; HEISEL Maritta (1998). An agenda for specifying software components with complex data models. In: *Computer Safety, Reliability and Security*, p. 16–31.

ZACHMAN, John A. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3), p. 276–292.

ZINNIKUS, Ingo; HAHN Christian; FISCHER Klaus (2008). A model-driven, agent-based approach for the integration of services into a collaborative business process. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, vol. 1. International Foundation for Autonomous Agents and Multiagent Systems, p. 241–248.

APIE TAIKOMŲJŲ PROGRAMŲ SĄVEIKUMO METODOLOGIJĄ, GRINDŽIAMĄ GILUMINĖMIS ŽINIOMIS

Andrius Valatavičius, Saulius Gudas

S a n t r a u k a

Įmonių taikomųjų programų sąveika dinamiškoje aplinkoje yra aktuali problema. Būtina ieškoti naujų metodologijų ir sprendimų. Siūlomo metodo metodologinis pagrindas yra vidinio modeliavimo paradigma, kuri integruota su MDA (OMG) metodu. Modifikuota MDA schema apima naują modeliavimo sluoksnį, skirtą žinioms apie realybės domeno savybes aprašyti, naudojami veiklos vidinio modeliavimo karkasai, grindžiami valdymo transakcijos konceptu. Modifikuota MDA schema leidžia apibrėžti organizacijos veiklos srities realybės priežastinius ryšius ir juos perduoti į skirtingus MDA sluoksnių modelius. Tyrimas remiasi prielaida, kad organizacijų veiklos sritis yra tikslo siekianti ir save valdanti sistema. Valdymo transakcija yra esminis veiklos valdymo vidinio modeliavimo konceptas, nes atskleidžia kiekvienos tikslo siekiančios veiklos vidines informacijos transformacijas (tai giliosios žinios apie save valdančias veiklas). Panaudoti veiklos vidinio modeliavimo karkasai leidžia atsekti realybės domeno – organizacijos veiklos – priežastines priklausomybes per visus programinės įrangos kūrimo MDA sluoksnius ir taip nustatyti domeno priežastingumo įtaką programos vientisumui ir sąveikai. Šis metodas jungia veiklos modeliavimo metodus ir reguliavimo teorijos principus, veiklos architektūros modeliavimo karkasus ir autonominio skaičiavimo koncepciją. Veiklos architektūros modeliavimo kalba ArchiMate yra vartojama priežastinių ryšių perdavimui tarp modelių, kurie yra skirtinguose MDA sluoksniuose, iliustruoti. Aprašyta šiuo metodu sukurta taikomųjų programų sąveikumą užtikrinanti programų sistemos architektūra su autonominiu integravimo komponentu.

2017 m. rugpjūčio 8 d.