Online ISSN: 2367-5357
DOI: 10.7546/itc-2019-0006

Events Proactive Management in Virtual-Physical Space

Z. Guglev, S. Stoyanov, I. Popchev, B. Toskov

Key Words: *Event model, virtual-physical space, event broker, virtual education space, JADE.*

Abstract. *This article describes improvements over the classical object-oriented solution for managing events in the Virtual-Physical Space, successor of the DeLC system, by introducing a higher-level implementation based on Java Agent Development Framework. The development brings proactive, agent-based solution for representation and distribution of events with abilities to interact with other FIPA rational agents.*

1. Introduction

Virtual-Physical Space (ViPS) [16] which is an extended version and successor of the Virtual Education Space [7, 20] and Distributed eLearning Centre [17] systems developed in the University of Plovdiv "Paisii Hilendarski" is built upon the Cyber-Physical-Social-Space and Internet of Things ideologies, utilizing different hardware and software layers to accomplish various purposes starting from simple environment monitoring to proactive personal assistance based on performance and behavioral analysis.

The latest version of the system resolves a major disadvantage in its predecessors – ignoring the presence of a physical environment which is often the source of many causal effects that will eventually propagate into the virtual space or vice versa. The majority of software operating inside the ViPS include ontologies, traditional software services, microservices, and rational agents, most of which operate in cooperative manner to achieve some common purposes. The specifics of the chosen distributed approach however bring their own group of problems that have to be solved in advance like for e.g. agreement upon common objectives, definition, execution and distribution of tasks, coordination of actions, data sharing, and state monitoring. The ViPS attempts to firstly solve those sub-problems by introducing its own event representation model and a set of algorithms and mechanisms for basic analysis, classification, distribution and storage, combining them and the model into a custom event engine.

Modern aircrafts have a wide variety of sensors that generate large volumes of data for different work characteristics. This data belongs to the so called Industrial Big Data (IBD) – big data arrays collected from any industrial equipment. The collection and storage of IBD in aircraft is at a local level. The technology allows online monitoring of the status of the equipment, which is expensive and its implementation is very selective. Current developments for this model include wireless sensors, a web browser that monitors the equipment status and an online alert system that informs the operator or the support team of any deviations. The information is sent by e-mail or text messages.

The following section gives more information background and ranges over approaches used in some of the related works while the other subsequent sections focus on describing the techniques used to transform the existing ViPS's event engine from an object-oriented to agent-oriented one, explaining the benefits of such an approach.

2. Existing theories and work

The concept for using events as a basic organizational structure or a way to exchange data is very common. Essential event forming and event processing approaches are described in [5]. Events are used to organize and access dynamic multimedia systems in [21], where similar events are differentiated from each other based on their spatiotemporal parameters and other specific to the event characteristics. Data-driven event triggering and low-level event-based communication techniques applied in IoT environments are described in [10]. Further details over the various basic event aspects are given in [9] – spatial, temporal, causal, structural, experimental, and informative while [14] explains ways to describe event flows and transitions in calculus form. In [13] is shown an alternative data compression and representation technique of event's embedded key performance indicators, the values of which in that case come from IoT devices. The event approach got included in OWL language as well, getting extended by [11]

with elements allowing modelling of event flows. Modular event architecture "MAIA" [15] suggests event based-approach for agent communication relying on JSON-LD as a format for data exchange backed by Event Web Ontology.

Event Engine 1.0

The current version of the ViPS includes an object-oriented implementation of event model and engine used to standardize representation and manipulation of events, the occurrence of which affect Space's domain and operation [4]. Although in its current form the model is flexible enough and relatively easy to integrate with most Space's components, a certain majority of units operating on higher abstraction level can benefit from a similar implementation based on the idea of human's practical reasoning, implemented as FIPA-compliant rational agents and an underlying group of behaviors.

Defining "Event"

The definition "event" has a great importance for systems operating with occurrences. The earliest version of the event model defines "event" concept as a basic principle and structure to organize, access and synchronize various dynamic systems within the ViPS. From the suggested in [4] options it is accepted the following: "something that happens or is regarded as happening; an occurrence, especially one of some importance; the outcome, issue, or result of anything; something that occurs in a certain place during a particular interval of time". Due to the broad range of definitions the term "event" can have, for the purposes of the ViPS it is preferred as a phenomenon occurring (or accepted as having occurred) within a particular location and time interval, the effect of which affects ViPS's operation. In other words, it is accepted that there are many kinds of events, but acknowledged and considered are only those affecting the ViPS.

Concepts for event representation

Because events can occur both in the physical or virtual world it is accepted differentiation based on this criterion. Separately, a way for symbolic representation in the virtual world needs to exist (including for virtualization of physical events). Finding the required, suitable representation for the purpose is a challenging task. The large variety of events with their characteristics and the fact that events depend on different factors like domain of interest, context, and granularity makes possible using alternative approaches to define them. One variant is the "bottom up" approach which aims for simpler representation, where some events are defined as more complex structures built out of other elementary events. Another possibility is reverse of the "top down" approach attempting to characterize an event in more detailed way by using different attributes.

For instance in [9] an event model called E-Model is proposed, introducing the following aspects of any event: temporal, spatial, informational, causal, structural, and experiential. In [12], for event it is stated "a thing happening in a certain time period and place, in which some actors participate and show some action features, along with the changing of internal status". Formally, an event is defined as a 6-tuple $\langle A, O, T, P, S, L\rangle$ where A means an action or a set of actions happening in event, O means objects involved in the event, T is the event's duration, P stands for the location of event's happening, S gives object statuses during an event happens, and L indicates language expressions of text-based event.

It can be seen two commonly preferred approaches to represent events:

- Atomic events – on lower level the events are represented as atomic structures without parameters. Complex events are built using the atomic ones.
- Attributed events – occupy a higher abstraction level, where every event is characterized by different attributes.

While the existing event representation approaches usually favor one of the two aforementioned methods, Space's model relies on a hybrid approach which allows different components to operate and work with different event aspects.

In Event Model 1.0 is accepted that E is the set of events happening within the ViPS and e is event such as

$$(1)\quad e = < d, y, a >,$$

with founding characteristics like fictive identifier d, event type y, and attributes a. The event e' is attributed if:

$$(2)\quad e' \in a(e).$$

The actual event representation can be done using recursive structure. Respectively e is an attributed event.

Let's have two events e', $e \in E$ such as e' is attributing and e – attributed. It is defined the following two operations:

- $e' \uparrow e(fire)$ – occurrence of e' causes the happening of e;
- $e' \downarrow e(kill)$ – occurrence of e' terminates the event e.

Let's have the two events e', $e \in E$ and define the following terms:

- $e' \parallel e$ (independent events) – the event e' does not "know" about the event e;
- $e' \rightarrow e$ (dependent events) – event e' premises the event e, in other words they are causal linked.

The event model supports classification like the following one:

(3) $E = BE \cup SE \cup DE.$

The three disjunctive sets above are:

- *BE* – the set of basic events, actual (time(Date, Hour), location);
- *SE* – subset of system events;
- *DE* – set of domain-dependent events.

Alternatively, if objectify at the most scalar level every event is assumed as a set of attributes. Time (*t*) and location (*l*) can form event's basic spatiotemporal identity, but their use is optional. As mentioned earlier there is an attribute (payload) section that might contain zero or more preliminary defined attributes as well as freeform ones. The section can be also treated as set that will be called *P*. For instance there's an event E_x with some attributes like:

(4) $E_x = \{ d, t, l, p_1, p_2, \dots, p_n \}.$

The payload section permits unlimited amount of scalar values along with recursive inclusion of other event definitions called sub-events that must represent proper subsets:

(5) $P = \{ p_1, p_2, p_n, \{d_n, t_n, l_n, \{p_n, \dots\}\} \dots \},$
$P \subset E_x \vee P = \emptyset.$

The aggregation of all definitions and their concrete values provides a uniquely identified complex event *C* preceded by other events becoming sub-events in its definition:

(6) $C = \big(A \cup \big(B \cup (\dots \cup Z)\big) \dots\big).$

The possible complexity is moreover limited from the model, by allowing only basic or system events as sub-events.

Furthermore, basic primitives are provided through which it is possible to execute comparison operations between simple or complex events. This can be done by specifying the individual members relative to which the comparison will be done. For example a simple comparison of two concrete events *X* and *Y* having a common integer field designating priority *u* can be done by referring to it (by its alias defined in the model) and the expected outcome *M* of the comparison:

(7) $X = \{ u \in \mathbb{Z} \},$
$Y = \{ u \in \mathbb{Z} \},$
$M = \{ 0 \},$
$Z = X \times Y = \{ (x, y) \mid x \in X, y \in Y \},$

(8) $S = \{-1, 0, 1 \},$

(9) $f: Z \longrightarrow A, \ A = \{ a \in S \}, \ |A| = 1,$

where

$$f(x, y) = \begin{cases} -1, & x < y \\ 0, & x = y. \\ 1, & x > y \end{cases}$$

The result can be directly compared to the expected outcome which will indicate whether the desired conditions are met or not. In case of complex comparisons including sub-events, *M* has to be adjusted accordingly to match the structure of the source data, while f will be applied seamlessly to the members that need to be compared. Multiple events or their particular attributes can be combined through the use of logical operations like conjunction, disjunction, and negation. The above approximation does not show handling of any special situations that might occur due to the architectural and implementation specifics of the model, although they are realized in it using Java programming language. Such cases are for instance broken reflexivity from *X* to *Y*, from *Y* to *X*, different and/or incomparable data types. Those cases are resolved in the model by assigning them special finite values [4]:

NOTCOMPARED = 2,
UNKNOWN = 4,
INCOMPARABLE = 8.

Additionally, the event model defines basic categories and hierarchy of events. Logically organized, the domain events take the highest place in the hierarchy, followed by system events, and lastly the basic events. In contrast, the programmatic implementation is done in nearly reverse approach, relying on object-oriented inheritance techniques. The different events and event categories retain specific properties and data organizing structures. Mechanisms for serialization and transportation via different broker messaging systems complement the model, forming the first version of ViPS's event engine [18].

Implementation drawbacks of Even Engine 1.0

Most of the components operating in the ViPS are implemented as rational agents [17]. While they can directly use the developed event model to communicate through events, the object-oriented nature of the engine and its mechanisms for event distribution in particular deviate the "agent" nature to "service-based" one. In result the agents are obliged to manually query through event engine at some interval the message broker system that's effectively delivering events. If new events are available the agent needs to decide which to ignore, analyse or react to accordingly and perhaps send back results by generating another event and forwarding it to the broker. All of the aforementioned steps need to be developed separately for the individual agent while considering that they have to be multiplexed with the inter-agent communication which

happens all the time [3, 19]. Possibility for important events to be able to interrupt the agent's communication or vice versa shall be also considered. Implementing the above requirements is usually a time- consuming process that is not very straightforward and requires careful planning of agent's architecture and apportioning its computational resources.

3. Developing Event Engine 2.0

The second version of the engine is based on its predecessor with the idea of keeping the existing representation model while improving the event distribution mechanism, making it more natural for use by rational agents and thus reduce their development time and complexity. The implementation is again based on Java programming language and relies in particular on Java Agent Development Framework [19]. The final result is a library containing set of behaviors, configuration utilities, communication protocols, ontology, and default realization of rational agent with event brokering functionality.

3.1. Proactive management of events

Instead of making every agent to check for new events, version 2.0 of the engine reverses the process by making new events to inform the interested sides for their existence. When the new event occurs, it is represented by own agent which announces its existence to the other agents in the system. With the increasing number of events however, this approach is rather inefficient even in environments highly optimized to support many agent instances. Instead, a single agent was designed to represent whole category of events. Furthermore, the event model permits event categories to inherit each other which allows filtering and reduction of the communication traffic only to those categories a particular side is interested in. The more general category is selected, the more events are going to be received (see *figure 1*).

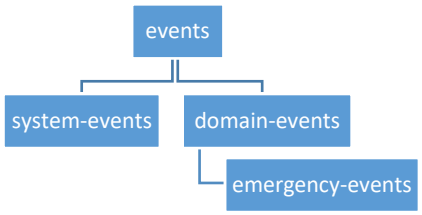


Figure 1. Object-oriented archetypal hierarchy and categorization of events

3.2. Event Engine for JADE

The developed solution is intended to be used as a Java library or on its own. In the former case the library gets included as part of the JADE agent that's being developed, providing it with set of components for solving various problems. The components can be used directly after specifying the required configuration parameters, but also allow further modifications and extensions. In the latter case the solution brings default implementation of a configurable event broker agent, through which bidirectional event exchange can happen.

3.3. Event engine for JADE library

The most important features brought by the library are set of behaviors automating event exchange process as well as ontology, definitions and implementations of communication protocols to assure proper data transfer. Every agent with ability to produce or consume events is required to support the "event-engine-ontology". Agent brokers announce themselves into JADE's Directory Facilitator service as supporting "event-engine-ontology" and by doing so they must be able to work with "FIPA-request", "FIPA-subscribe", "event-engine-ping" and "event-engine-channel-event-exchange" communication protocols. The first two protocols are well-known and standardized while the rest are specially implemented for engine's purposes. Implicitly for the end user the library also relies on "FIPA-Agent-Management" and "JADE-Agent-Management" ontologies and their underlying protocols, already implemented in JADE [3, 6, 19].

From user agent's perspective the only required actions to make it able to send and receive events is instantiating the provided by the engine behaviors for search and subscribe to event broker agent, and the one creating channel for event data exchanging.

3.4. Searching for event broker agents

The term "event broker agent" is used for special type of rational agents that connect to traditional message broker systems used by the event engine for distribution of events. Usually many broker agents are expected to exist, each one of them responsible for a particular event category with certain capacity of how many clients it can handle. The agent brokers communicate with each other using message broker systems, while communication with ordinary agents is done via agent communication language based on the FIPA-SL specification.

Event Engine 2.0 (see *figure 2*) provides behavior allowing seamless, automated searching and subscribing to event broker agents, based on certain criteria like for e.g. event category and agent proximity. The behavior takes into account if the broker's capacity is reached or the broker is not responsive due to overloading or connectivity problems. In such cases the behavior will automatically try to find another suitable broker and if such is found new subscription will be performed, while unsubscribing from the older one.

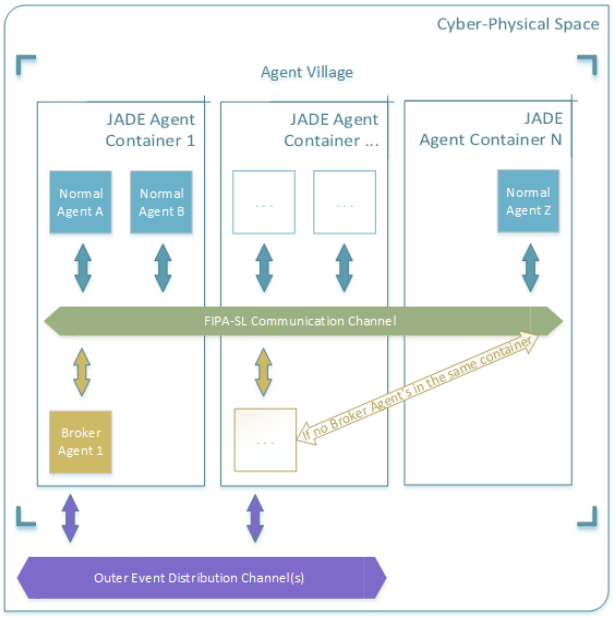


Figure 2. Configurable event broker agent(s) use case architecture

3.5. Event exchanging

Event engine includes a separate behavior through which ordinary and event broker agents can send and receive events. Depending on which agent initializes it a data transfer channel to message broker agent or message broker system gets created. Sending and receiving events through the channel is asynchronous, placing them in an internal queue for further processing. This allows manipulation of the data in batches and optimizes the computational process. Upon receiving events, the agent is either allowed to either "subscribe" its own methods for automatic execution against the received data (similar to "observer" design pattern) or to manually check for new events and decide how to process them. In addition to the above, the behavior automatically converts event instances from the event model from and to ACL FIPA-SL compatible messages, verifies that the client side has fully received the sent data, and also handles cases when event cannot be sent due to broker unavailability. The last functionality can also work in sync with the behavior for automatic search and subscribe for broker agents.

3.6. Event broker agent

The main purpose of engine's event broker agent is to provide communication node with characteristics of event dispatcher and proxy. For every successfully subscribed agent client the broker creates an internal connection to an actual message broker system. Any ACL message containing events addressed to the broker by another agent will be passed through the corresponding message broker connection. Receiving events from the message broker connection leads to converting them into appropriate ACL messages, several of which might be accumulated into one data packet unit that will be sent to the corresponding agent. The event broker agent requires to be provided with a configuration file upon execution, containing variety of settings for controlling different behavioral aspects like for e.g. used data encoding mechanism when sending events through message broker system, data dispatching mode, the event category to work with, broker system credentials, subscribers limit, how often to dispatch data, how often to check subscriber's availability, etc. Some of the settings are automatically remapped for every client agent initiating subscription, which makes possible to uniquely identify itself in front of the end broker system, to use persisting and/or retroactive data retrieval techniques.

Another important aspect is broker agent's ability to simultaneously work with several message broker instances that might be of the same or different types (see *figure 3*). This makes possible for the agent to perform event relaying between isolated broker nodes running different broker instances or entirely different broker systems. Currently the engine supports Apache ActiveMQ and Apache Kafka systems [1, 2].

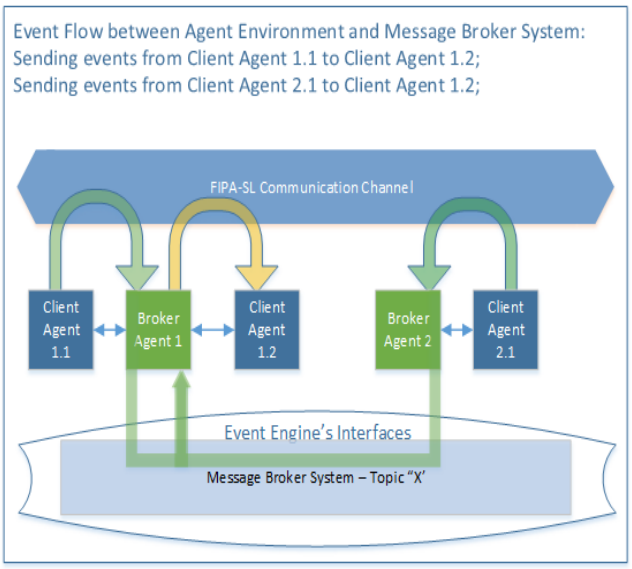


Figure 3. Data exchange and relations between agents, event broker agents, and message broker systems

3.7. Event preparation before exchange

To comply both with FIPA-ACL, ActiveMQ and Kafka standards the event exchange process passes the data through several transformations to ensure unified, protocol-friendly stream of bytes that's not interfering with other exchange protocols. The first step of the process includes normal serialization from in-memory Java objects to finite sequence of bytes. The actual format might be JOSS or JSON. To overcome any interference that might happen

with FIPA-ACL protocol and its limited subset of characters and rules that apply to them additional BASE32 encoding [8] is executed on top of it. BASE32 has a lower encoding efficiency of around 8:5, but provides a relatively save, limited set of encoding characters. The results are then placed into a custom data packet structure, intended for sending through message broker systems, that combines the encoded bytes with metadata describing packet's potential contents. This gives the receiver the opportunity for preliminary packet inspection, deciding whether the contents of the packet are supported or not and possibly discard them if the second case is in effect. The described so far process might end by sending the packet through the underlying message broker systems if the sender turns out to be event broker agent or non-agent-based component. In case the sender is an ordinary agent, the prepared data packet(s) are accumulated into JADE ontological concept structure called "EventData". The concept is used during inter-agent communication and seamlessly gets converted from and to FIPA-ACL format (see *figure 4*). Its ability to contain multiple actual data packets drastically reduces the number of messages exchanged between the agents and thus lowers the load on the systems. The exact same process described above is executed in reverse when agent receives event data.

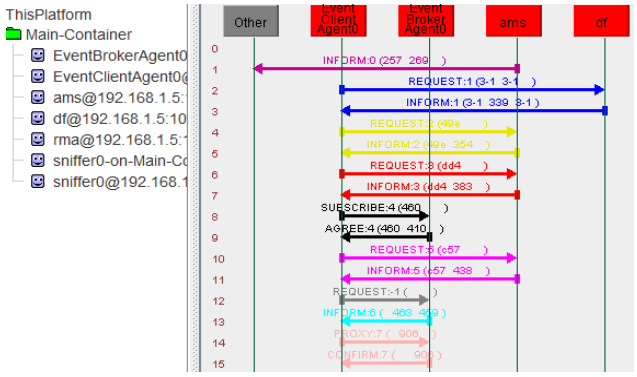


Figure 4. Sample communication between client and agent broker

3.8. Deduplication of events

Using more than one message broker system by single event broker agent results any ACL messages sent to the broker to be forwarded to all of its message broker systems. Without further processing any other agents subscribed to the same broker agent will receive the exact same message multiplied by the number of message broker system instances. To prevent such effects two-side data deduplication is attempted in the event channel behavior. The deduplication is based on distinguishing via dynamically appended to the particular event identification data, just before being sent through the message brokers. Upon receiving and before passing the event to the end destination that information is checked against a non-persistent collection containing identity data of the latest received events. A match will indicate that the event has already been processed so it is safe for the current copy to be discarded. The technique however is limited by the storage capacity of the collection. In scenario where one message broker system receives far greater amount of data compared to another one a possibility for the deduplication information to be thrown out too soon exists. For that reason similar deduplication checks are executed on both event broker and subscriber agents. Nevertheless this still does not guarantee entirely duplicates-free event flow, which suggests careful consideration before deciding to use event relaying. If this is necessary, and depending on the case, the default channel behavior might be modified to use persisting event deduplication data collection.

4. Conclusion

Extending the existing object-oriented event engine to an agent-based one allowed proactive management of events within the Virtual-Physical Space. By providing a separate broker agent to represent concrete event category resulted simpler, robust and traffic-optimized event exchange process. In addition to that the second engine version made possible for the other agents not to differentiate between natural agent-to-agent, agent-to-broker, and vice versa communication. As a result, the development complexity and time for ViPS's agents to achieve "general" event-oriented behavior got significantly reduced. Since the new engine is based on the data representation model and distribution techniques of its predecessor, a full compatibility is maintained with the solutions using its older version.

Acknowledgments

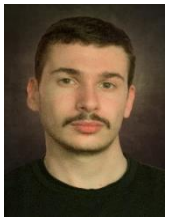
The research is partly supported by the MES by the Grant No. D01-221/03.12.2018 for NCDSC – part of the Bulgarian National Roadmap on RIs.

References

1. Apache Foundation, "Kafka Documentation", available at https://kafka.apache.org/documentation/, last accessed 2020/05/20.
2. Apache Foundation, Apache ActiveMQ, available at http://activemq.apache.org/, last accessed 2020/05/20.
3. Bellifemine, F. L., G. Caire, D. Greenwood. Developing Multi-agent Systems with JADE, John Wiley & Sons, Inc., 2007.
4. Dictionary.com, "Definition of event", available at http://www.dictionary.com/browse/event?s=t, last accessed 2020/05/20.
5. Etzion, O., P. Niblett. Event Processing in Action, Greenwich: Manning Publications, 2011.
6. Foundation for Intelligent Physical Agents (FIPA), Standard Status Specifications, available at

http://www.fipa.org/repository/standardspecs.html, last accessed 2020/05/20.
7. Gramatova, K., S. Stoyanov, I. Popchev. Virtual Education Space Realization as an Internet of Things Ecosystem. – *Engineering Science*, LV, 2018, 1/2018, 5–19.
8. IETF, The Base16, Base32, and Base64 Data Encodings, October 2006, available at https://tools.ietf.org/html/rfc4648, last accessed 2020/05/20.
9. Jain, R. EventWeb: Developing a Human-Centered Computing System. – *Computer*, 41, 2008, 2, 42–50.
10. Kolios, P., C. Panayiotou, G. Ellinas, M. Polycarpou. Event-Based Communication for IoT Networking. Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 2015, doi: 10.1109/WF-IoT.2015.7389076.
11. Liu, W. Z. Liu, J. Fu, R. Hu, Z. Zhong. Extending OWL for Modeling Event-oriented Ontology, Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 2010, doi: 10.1109/CISIS.2010.88.
12. Liu, W., Y. Tan, N. Ding, Y. Zhang, Z. Liu. An Ontology Pattern for Emergency Event Modeling, Proceedings of the 2016 IEEE 14th Int Conf on Dependable, Autonomic and Secure Computing, 14th Int Conf on Pervasive Intelligence and Computing, 2nd Int Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech, Auckland, New Zealand, 2016, doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.44.
13. Moawad, A., T. Hartmann, F. Fouquet, G. Nain, J. Klein, Y. L. Traon. Beyond Discrete Modeling: A Continuous and Efficient Model for IoT, Proceedings of the 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), Ottawa, ON, Canada, 2015, doi: 10.1109/MODELS.2015.7338239.
14. Mueller E. T. IBM Watson Group and IBM Research, Commonsense Reasoning. An Event Calculus Based Approach, 2nd Ed., Morgan Kaufmann, 2015.
15. Rada, J. F. S., C. A. Iglesias, M. Coronado. MAIA: An Event-based Modular Architecture for Intelligent Agents, Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, 2014, doi: 10.1109/WI-IAT.2014.154.
16. Stoyanov, S., A. Stoyanova-Doycheva, T. Glushkova, E. Doychev. Virtual Physical Space – An Architecture Supporting Internet of Things Applications. Proceedings of the XX International Symposium on Electrical Apparatus and Technologies – SIELA 2018, Burgas, 2018, 251–252.
17. Stoyanov, S., I. Popchev, E. Doychev, D. Mitev, V. Valkanov, A. Stoyanova-Doycheva, V. Valkanova, I. Minov. DeLC Educational Portal. – *Cybernetics and Information Technologies*, 10, 2010, 3, 49–69.
18. Stoyanov, S., V. Valkanov, I. Popchev, A. Stoyanova-Doycheva, E. Doychev. A Model of Context-aware Agent Architecture. – *Comptes rendus de l'Académie bulgare des sciences*, 67, 2014, 4, 487–496.
19. Telecom Italia Lab, JADE v4.5.0 API, available at https://jade.tilab.com/doc/api/index.html, last accessed 2020/05/20.
20. Valkanov, V., S. Stoyanov, V. Valkanova. Virtual Education Space. – *Journal of Communication and Computer*, 13, 2016, 64–76.
21. Westermann, U., R. Jain. Toward a Common Event Model for Multimedia Applications. – *IEEE MultiMedia*, 14, 2007, 1, 19–29.

Manuscript received on 16.12.2019



Zhelyan Guglev *is a Ph.D. student in the University of Plovdiv "Paisii Hilendarski". He received his M.Sc. degree in Software Technologies from the University of Plovdiv in 2015. His current research interests are in the field of event modelling, event processing, intelligent software agents, and the Internet of Things*

Contacts:
Department of Intelligent Systems
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Sofia, Bulgaria
Department of Computer Systems
Plovdiv University "Paisii Hilendarski"
Plovdiv, Bulgaria
e-mail: stani@uni-plovdiv.net

Prof. Stanimir Stoyanov *is a Member of the Institute of Electrical and Electronic Engineers (IEEE), of the Association for Computing Machinery (ACM) and of Bulgarian Society of Automatics and Informatics. He received his doctoral degrees from the Humboldt University of Berlin, Germany and the De Montfort University of Leicester, UK. Currently, he is a Lecturer in the Plovdiv University "Paisii Hilendaski", Bulgaria, where he also acts as a Head of the Department of Computer Systems. His research interests include: intelligent agents, agent- and service-oriented architectures, CPSS, IoT, context-aware and adaptable software, eLearning tools and environments. Publications: 200+, 5 books.*

Contacts:
Department of Intelligent Systems
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Sofia, Bulgaria
Department of Computer Systems
Plovdiv University "Paisii Hilendarski"
Plovdiv, Bulgaria
e-mail: stani@uni-plovdiv.net

Ivan Popchev *since 2003 is a member of Bulgarian Academy of Sciences. He received Ph.D. degree in 1967, with a thesis "Studies on the performance index of large scale control systems" and D.Sc. in 1980, with a thesis "Multicriterial synthesis of control systems". He has more than 480 papers presented at world congresses, conferences, symposia and workshops and/or published in international journal/periodicals, proceedings and books in English, German, French, Russian, Korean and Bulgarian. His research interests are in the fields of computer science, mathematical modelling, decision sciences and control theory.*

Contacts:
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Sofia, Bulgaria
e-mail: ipopchev@iit.bas.bg

Bosrislav Toskov *is a Ph.D. student in the Plovdiv University "Paisii Hilendaski", Bulgaria. His research interests are focused on Future Networks and the Internet of Things with applications in sustainable and independent living: IoT, multi agents software systems, biometric identification, wireless M2M communication.*

Contacts:
Department of Computer Systems
Plovdiv University "Paisii Hilendarski"
236 Bulgaria Str., Plovdiv, Bulgaria
e-mail: btoskov@uni-plovdiv.bg