HYBRID APPROACH FOR MANIPULATION OF EVENTS IN THE VIRTUAL REFERENT SPACE

Zhelyan Guglev, **Stanimir Stoyanov**
Faculty of Mathematics and Informatics, University of Plovdiv „Paisii Hilendarski”
jelian_g@mail.bg, stani@uni-plovdiv.net

Abstract: *The referent model of the Virtual Education Space which is successor of the environment for electronic education DeLC relies on a diverse Internet of Things ecosystem built out of intelligent software and hardware components to analyse and interact with the surrounding environment. One method for them to communicate is by utilising an event-oriented approach. The various kinds of tasks, execution contexts and physical environments where the events can happen require the development of a flexible and portable model, combining different distribution and event processing methodologies described in this report.*

Key words: *Event model, event distribution, event engine, message broker system, Virtual Referent Space*

INTRODUCTION

Problems like the agreement upon objectives, definition of roles, coordination of actions, distribution and collaborative execution of tasks, sharing consistent data, and state monitoring are just part of the common technical challenges, the majority of distributed software systems needs to solve. The Virtual Education Space as such system is relying on the Internet of Things ideology, utilising variety of hardware and software layers providing ontologies plus traditional, micro and intelligent (based on practical humans' reasoning model) services is not making an exception from the above statement [1]. One approach that has been taken in the direction of solving the aforementioned problems is to represent any occurrences happening within the domain of the Space and its subsystems using an event-oriented approach, the concept of which has been originally introduced in [2] while some of the earlier use cases have been described in [3].

A common model for representation and distribution of events is required by the components of the Virtual Education Space in order for them to be able to achieve an event-oriented behaviour. The implementation of the model needs to be flexible enough to represent different aspects that sometimes might be absent from the occurring events, but also needs to provide mechanisms for event manipulation like basic analysis, classification, distribution, routing, and storage.

While the architectural and implementation details of model for representation and basic differentiation and analysis of events is described in [4], the problems regarding the actual distribution and initial post-processing mechanics still had to be solved. The distribution process needs to be reliable enough in order for the data to be received from all interested sides while the implementation has to provide flexibility and ease of use for components operating both on traditional, rich of resources computer systems, and on embedded (often IoT) devices.

EVENTS AND EVENTS-ACCEPTED SYSTEMS

The concept of „event“ serves as a basic organizational principle as a basic structure for organizing and accessing dynamic multimedia systems [5]. Usually, the main features of an event are identification, time and location; the latter two becoming the basic defining characteristics of the event. Thus, similar events occurring at different time and/or at a different place in space are considered to be different. In this sense, an event is defined in spatial-temporal dimensions. An event can be map various aspects such as temporal, spatial, causal, structural, informational, experimental [6]. Actually, events-accepted systems use event ontologies; to create an event ontology an extended version of the OWL language was proposed in [7]. Also, an event description logic is proposed which provides a logical basis for OWL extensions. In line with the new trends in web programming [8] presents the agent-oriented architecture known as MAIA that supports an events-oriented work. In particular, this architecture focuses on personal agents that interact with different web services. With platforms like MAIA it's easier to use agents in a Web Event providing various services. For example, in [9] an alternative way of identifying the curriculum is proposed by analogy with complex event processing and event-oriented architecture. A modeling method called EDUMO is also proposed in this paper. In the form of a calculus, a formal event model is proposed in [10].

MATHERIALS AND METHODS

Categorisation of events. To perform efficient distribution of events the generated data traffic needs to be optimised, otherwise it might cause communication congestions, delays, overwhelm and even block the sides that do not have enough processing power. In large distributed environments, with limited control over the individual components that can have complex architecture and interrelationships, very often this constrains the amount of effective actions that can be safely applied.

The taken approach was to introduce a way to group similar event types by assigning them a category while delegating the responsibility for controlling the amount of generated traffic to the individual event producers. The underlying event definitions in the original event model were retrofitted to contain a category section that gets inherited by any new implementers, but can be also changed.

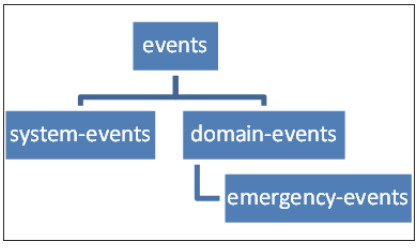

Fig. 1 Categorisation of events

As an additional effect the categories will diverge more the farther they get from their parents' definition, becoming more concrete. Finally the technique made possible for a particular side to monitor less events at once, giving it the ability to change the level of concreteness by ignoring whole unrelated categories and their more specific children.

Serialisation of events. Due to the fact that the implementation of the event model is done entirely in Java it is mandatory requirement to exist a mechanism that would allow transformation of event class instances from the operating memory to an efficient discrete representation format, suitable for transfer through alternative mediums like for e.g. TCP/IP networks. Performing even the most basic event distribution without such mechanism is not possible, thus the following formats were introduced:

- Java Object Serialization Specification – the standard, internal serialisation format supported by Java, that is easy to use, widely spread, easy for achieving interoperability with other existing Java components and services, but limiting the class's flexibility once implemented as well as increasing the likelihood for introduced bugs and security holes. The data representation format although well documented is binary, which makes it harder to read by humans and harder to be implemented in non-Java platforms [11];
- JavaScript Object Notation – widely spread including outside of Java's ecosystem, represented in human-readable text format, less limiting the class's flexibility, with lower likelihood for bugs and security holes introduction, simpler to implement [11], used even by IoT devices;
- Base32, RFC 4648 – not an actual serialisation format, but an encoder using limited set of text symbols to represent the original data. For event distribution it needs to be combined with an actual serialisation format over which the encoding to be applied. It was found to be particularly useful in FIPA-compliant multi-agent systems as a safe way for direct insertion of event data in agents' communication stream without corrupting the text-based agent conversation. Despite the fact that Base32 is using limited alphabetical set, which makes it less efficient in data representation with ratio of approximately 8:5, the benefits of its use in agents' communication were found to be greater.

Packaging events. Providing a standardised event distribution mechanism requires encapsulation of the serialised data in a well-known format, called in event model's terms „data packet". The packet's purpose is to provide unambiguous, universal, brief description of its data contents like for e.g. the used serialization mechanism and encoding. The format needs to be simple enough and platform independent so even less powerful IoT devices to be able to decode it on its own and should also maintain a backwards compatibility via some kind of versioning information in case of future changes. The following simplified and partially text-friendly format was designed.

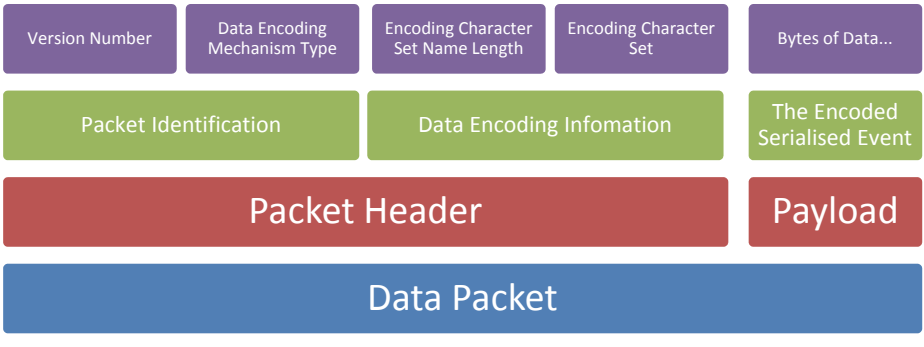

Fig. 2. Data packet for encapsulation of events

As illustrated on ***Грешка! Източникът на препратката не е намерен.*** the data packet starts with a version number represented with a single byte. The next following byte stands for the used data encoding mechanism, where several values are reserved including one for designation of unknown encoding technique. The third byte describes the length in bytes of the name of the character set that is used for representation of the payload. This provides compatibility between the different systems and platforms, where the programmer has freedom to pick the most suitable set of characters, as long as the receiving side

supports it. The default choice is set to the common ISO-8859-1 standard to minimize the size of the encoded data while maximizing the chances for the opposite side to be able to decode it. The name of the standard is also encoded using ISO-8859-1. Last in the parameter chain follows the payload represented again using bytes. There is no need for description of its length, because the size can be determined when the end of the packet is reached, or alternatively by subtracting from the whole packet size (which is known, automatically provided by the receiving system) the size of the packet header.

Distribution of events. The whole process of sending event model's events, receiving those sent from the opposite side, and the ability for the receiver to „subscribe" itself only to those event categories it is interested in is described as „distribution of events", backed by the intrinsic functionality of the model. The combination of the basic (described in [4]) and distribution functionality is defined as „event engine" in the referent model of the Virtual Education Space.

To distribute events the engine relies on well-established message broker systems, taking advantage of the „publish and subscribe" semantics and in particular their broadcast-based variation. The approach defines information subjects called „topics" that contain a certain type of data [12]. In the even engine's case every „event category" relates to a single broker „topic". Based on the category of the instantiated event, the data gets automatically converted to „data packet" and then forwarded to its proper destination. It is a very common for the most message broker systems operating in broadcast-based regime to provide a fixed amount of topics. In event model's case however this is not flexible enough, because new event categories might emerge in the future, so as a basic requirement for any broker system in order to be compatible with the model is to allow dynamic creation of new topics.

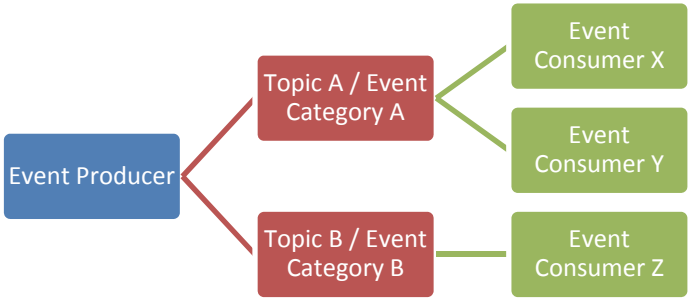

Fig. 3. Sample event distribution using topics

Although the software developer can directly use the public interface of a particular message broker to distribute events, there will be implementation challenges to overcome, and while solving them the level of abstraction of the end product might significantly decrease. Such problems are for e.g. the fact that a data producer is not a consumer (Fig. 3). However, very often in the Virtual Referent Space event producers are interested in receiving other events, the category of which might or might not match with what they are sending. This means that for the manual use of any broker in such scenario the programmer needs to handle three different cases – sending events, receiving events, and exclusion of duplicated events. The implementation details can distract the developer even further if at

some later point the distribution needs to happen through another or multiple broker systems.

The event engine provides a simplified, higher abstraction as a solution to the above problems, hiding big part of the technical details as well as the major ideological broker differences. In addition it defines and automatically manages three own, high level distribution modes – consumer, producer, and consumer-producer. This is accomplished through the use of a common facade technique combined with several abstract, concrete factories and proxies. The current implementation provides support for two fundamentally different systems - Apache ActiveMQ which is a traditional message broker system and Apache Kafka which is a distributed streaming platform that can also work as an enterprise messaging system [13]. That exact choice for initial support of those two systems (although the engine can be easily extended to support more) enables the developer to pick the most suitable one for the problem that has to be solved – event broadcasting, distributed event processing, or even a hybrid of both with real-time data transformations in between.

Retroactivity and persistence. Event engine's flexible broker support implementation allows its clients to interact with the remote brokers in retroactive and non-retroactive way. The retroactivity gives particular side ability to receive data originating before the time point of side's subscription. That feature can be especially useful to any Virtual Referent Space's components with rational behaviour analysing current and older data. Respectively any reactive components interested only in events happening right now can take advantage of the non-retroactive mode which will limit the data from the time of their subscription.

In addition to the above, clients may choose whether to be remembered by the broker or not. The „remembering" happens by supplying a unique identifier, which by default the event engine generates automatically, but also allows to be overwritten. In result for any client that has chosen to be remembered, the event data will persist during its absence and will be later streamed back when becomes online. Respectively any clients using non-persistent mode will not receive any events sent during their offline period. The reliability of the above features nonetheless depend on variety of factors like for e.g. broker's type, retention period settings, available disk space, maximum allowed connections and so on.

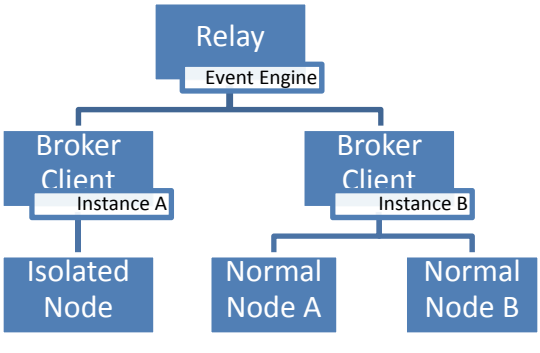

Fig. 4. Event relaying

Event relaying possibilities. The traditional case for event distribution can be described with multiple event producers and consumers that exchange data through the same medium (usually message bus). The architecture of the Virtual Referent Space however permits the existence of nodes that run in isolated environments with their own technical specifics. The ability of the event engine to allow simultaneous use of multiple message broker instances of same or different type can be used for event relaying between such isolated nodes as long as the „relay” has access to all of them.

Once the broker connections have been established every node is able send data that will certainly reach the entities within the same node or neighbour nodes sharing the same broker system, and eventually any other nodes that use different broker system might be also reached through the „relay”.

DISCUSSION

The distribution mechanism testing was done using engine’s „consumer-producer” mode in environment consisting of several rational agents as well as a couple of reactive services. The experiments were performed using two separate broker instances that were kept decoupled for the whole time. The first broker instance was running Apache Kafka in messaging system mode, while the second instance was using ActiveMQ. Leaving out the fact that the two systems at the end of the testing contained different data sets, the switching between them back and forth gave consistent results in terms of messaging experience. The responsiveness was found to be adequate enough with instant notification of the consumers when an event was generated by the producers. The resource utilisation on ActiveMQ was found to be 1.5 to 3.5 times greater compared to Kafka due ideological differences and the way they are handled by the event engine. Because of that improvements in the direction of supporting „ActiveMQ Artemis” variation with better data routing capabilities might be considered in the future. However if distributed event processing is needed, Kafka still remains a better choice.

CONSLUSION

The developed mechanisms for event distribution, data post-representation and seamless conversion between different forms provide higher level of abstraction, portability and simplicity over the direct use of message broker systems. While the event engine and its event distribution capabilities can be directly used by traditional Java-enabled systems, the chosen intermediate format is simple enough to be used by alternative components and services including IoT devices. Combed with existing solutions like „Kafka REST Proxy” [14] or the provided by ActiveMQ „Message Servlet” and support for protocols like AMQP, MQTT, REST, and Stomp [15] the event model can be directly used as the de facto communication standard between Virtual Referent Space’s IoT ecosystem and its other components.

Acknowledgments. The research is partly supported by the the NPD - Plovdiv University under Grant No. MU17-FMI-001 „EXPERT (Experimental Personal Robots That Learn)”, 2017-18.

REFERENCES

1. V. Valkanov, S. Stoyanov and V. Valkanova, „Virtual Education Space“, *Journal of Communication and Computer*, no. 13, pp. 64-76, 2016.
2. D. Orozova, S. Stoyanov, and I. Popchev. 2013. Virtual Education Space. In International Research Conference Knowledge- traditions, innovations, perspectives. 153-159.
3. S. Stoyanov, V. Vladimir, P. Ivan, S.-D. Asya and D. Emil, „A model of context-aware agent architecture“, in *Comptes rendus de l'Académie bulgare des sciences: sciences mathématiques et naturelles*, 2014.
4. Z. Guglev и S. Stoyanov, „Модел за представяне и разпространение на събития във Виртуалното Образователно Пространство“, в *International Conference „John Atanasov“*, Sofia, 2017.
5. R. Jain, EventWeb: Developing a Human-Centered Computing System, Computer, February 2008.
6. U. Westermann, R. Jain, Toward a Common Event Model for Multimedia Applications, IEEE Multimedia, January-March, 2007.
7. W. Liu, Z. Liu, J. Fu, R. Hu, Z. Zhong, Extending OWL for Modeling Event-oriented Ontology, 2010 International Conference on Complex, Intelligent and Software Intensive Systems, 581-586.
8. J. F. Sanchez-Rada, C. A. Iglesias, M. Coronado, MAIA: An Event-based Modular Architecture for Intelligent Agents, 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 87-94.
9. J. Lang, V. Krajčovič, Curriculum as an event hierarchy model, 9th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA 2011), Slovakia, 127-130.
10. T. E. Mueller, IBM Watson Group and IBM Research, Commonsense Reasoning An Event Calculus Based Approach, 2nd ред., Morgan Kaufmann, 2015.
11. J. Bloch, Effective Java, 3rd ed., Addison-Wesley, 2018.
12. Microsoft, „Publish / Subscribe“, Microsoft, 2004.
13. Apache Foundation, „Kafka Documentation“, [Online]. Available: https://kafka.apache.org/documentation/. [Accessed 08 May 2018].
14. Confluent.io, „Confluent Platform“, [Online]. Available: https://www.confluent.io/. [Accessed 13 May 2018].
15. Apache Foundation, „Apache ActiveMQ“, [Online]. Available: http://activemq.apache.org/. [Accessed 13 May 2018].