10.1515/acsc-2017-0018

Archives of Control Sciences
Volume 27(LXIII), 2017
No. 2*, pages 279–291*

Active resources concept of computation for enterprise software

MACIEJ KORYL

Traditional computational models for enterprise software are still to a great extent centralized. However, rapid growing of modern computation techniques and frameworks causes that contemporary software becomes more and more distributed. Towards development of new complete and coherent solution for distributed enterprise software construction, synthesis of three well-grounded concepts is proposed: Domain-Driven Design technique of software engineering, REST architectural style and actor model of computation. As a result new resources-based framework arises, which after first cases of use seems to be useful and worthy of further research.

Key words: domain-driven design, REST, actor model.

1. Introduction

Enterprise software systems working in deployment environment of huge corporations such as banks or industrial plants consist of many separate products, typically from several to several dozen parts. One software product serves from hundreds to above thousand use cases and at the same time interacts with use cases of other products due to automation of complex business processes. As regards the computation character, two types of processes may be listed:

- processes of interactive character, often automated by usage of workflow tools, responsible for entire process composition from atom elements representing well defined and cohesive activities. Process composition in such manner is called orchestration;
- processes of batch character, consisted of processing fragments one following another or one running parallel with another, typically iterated on collections of business objects characteristic for particular area, such as contracts, transactions, orders, etc. Nowadays implementations of such batch processes are supported by modern software frameworks dedicated to that purpose.

The Author is with Rzeszow University of Technology, W. Pola str. 2, 35-959 Rzeszow, Poland, e-mail: maciej.koryl@gmail.com

Received 18.12.2016.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

In both cases, constructed processes consist of many functions working in close cooperation, often exposed as APIs of systems or APIs of their components. As long as cooperation is carried in synchronous way, complexity of such arrangement may be controlled, even if number of involved systems is substantial and number of interactions is high. In synchronous systems number of possible states in which system may occur is countable and predictable at the stage of software designing or detectable during testing. After applying asynchronous model of communication, complexity of system violently grows with increase of possible different states, which number is non-linear function of possible states of constituents of the system and number of interactions between them ([4]). During many years of computation theories development and many years of software engineering implementation practices, meaningful conceptual and technical tools were established, but that does not mean that problem of complexity has passed away or even has been minimized to notable degree. The matter is broadly recognized in specialized computation areas dealing with well-established algorithms, but still is not enough captured in commercial products development, govern by its own specificity connected with high number of software users, many different business objects and huge number of unpredictable interactions (sample characteristic of such systems is shown in [8]). In these days, problem is more and more complicated because of limitations of monolithic systems and the need for introduction of distributed software, for example in the form of microservices ([12]), which are adapted for horizontal scaling and well suited in actual hardware capabilities. Several conceptual and technical tools currently used in software engineering discipline, dedicated to distributed processing are described in [3], where one of them is an actor model of computation, which after many years of academic development, currently gains great popularity in commercial area. Proposition which is shown in this paper constitutes coherent and complete framework based on well-tried techniques and design patterns with actor model between them and has working implementation in Java programming language with use of modern tools for enterprise applications such as Spring Framework and noSQL databases. Proposed solution has been used to build some parts of banking transactional system, which supports batch processes such as massive transactions processing or interaction with trade platform. As a foundation of the idea, three engineering concepts act: Domain-Driven Design technique proposed in [5] and broadly accepted in software community, Representational State Transfer architectural style introduced in [6] and currently becoming the most popular way of interaction between web components, and the actor model of computation described in [10] nowadays gaining mature and useful implementations. In addition, the solution was enriched by use of standard language of agent communication in multi-agent systems – Agent Communication Language, which semantics was found as very suitable for required interactions.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

2. Fundamental concepts

2.1. Resource as a central point of the REST model

Representational State Transfer (REST) is an architectural style implementing fundamental rules of the web and HTTP standard. REST was introduced by dissertation [6] and at present has obtained great popularity as 'the web used correctly'. Central idea of the style is to treat all things which have identity as resources and give them globally unique Uniform Resource Identifier (URI). Resources named in this way may communicate together using hyperlinks. Communication is provided by usage of standard HTTP commands with their established semantics. Important rule of REST is that communication ought to be stateless, i.e. parties cannot keep state of communication assuming that next message will be continuation of previous one. In proposed approach the resource term plays key role as external representation of computation units and set of REST rules and good practices in interactions modeling are applied.

2.2. Aggregate in Domain-Driven Design concept

Domain-Driven Design (DDD) concept introduced in [5] is an approach to software development which pays attention to key meaning of domain model in software design. Domain model plays central role in whole process of development acting as universal medium of communication between all participants and providing stable base for software structure. DDD technique is divided into two stacks of patterns: strategic and tactical ones. First of them serves as toolset for taking control over complexity of extensive software and second consists of a set of building blocks, which is sufficient for complete design of each kind of enterprise software on some level of abstraction. The most important pattern from tactical stack is the aggregate building block and there is plenty of rules explained in literature, how to build useful aggregates (for example [18]). Aggregate is a graph of objects tied together into one coherent object offering common set of services for external world. The only way to access aggregate constituents capabilities is aggregate root, the central entry point to that software unit. Thanks to such construction, aggregate guarantees the consistency of changes of whole structure, controls its internal state and gives convenient way for its access. In proposed framework, aggregate plays important role as representation of stable state of a resource and also as a part of resource in dynamical state by offering its behavioral capabilities.

2.3. Actor model of computation

The actor model of computation developed many years ago and firstly published in [10] was thought as conceptual tool for understanding of concurrency. Many software frameworks based on actor model have been built to this day, but broad utilization in enterprise software area is still scarce. Currently, attention in that idea is growing, stimulated by development of multi-core processors and development of cloud computing solutions with necessity of computation distribution. Theory of actor model treats actors as universal primitives with capability to carry out each kind of needed computation

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

([9]). Actors are independent units of computation loosely coupled together, only by asynchronous messages passing and the only reliable knowledge which actor has about other actor is its mailbox address. When an actor receives a message it may do some computation, send messages to participants, create additional actors as its children or may change its own behavior preparing itself for future course of situation. In proposed framework, actors will support implementation of dynamical state of resource.

2.4. Agent Communication Language

Agent Communication Language (ACL) is a definition of standard language used in multi-agent systems to model conversations between involved parties. Its origins are in philosophical theory of speech acts ([15]), which state that each utterance has not only informative, but also performative function, i.e. causes consequences in receiver's activity. ACL has been drawn up by Foundation for Intelligent Physical Agents (FIPA) as FIPA-ACL set of standards [7]. In proposed solution ACL syntactics and semantics are used to model communication between resources, especially by use of standard vocabulary of performatives denoting the type of communicative acts.

3. Active resources model of computation

3.1. The active resources term

As 'active resources' are considered resources, which at moment of interaction may be under change originated from other interaction, computation processes or any other factor. As active resource is in unstable state and its properties may change in time, another object cannot assume that something is true about that resource, even if resource still exists or not. An observer cannot have knowledge about its state, but can have only some beliefs. For example, if some procedure in banking system completes payment and has information about sufficient balance of debited account, it cannot assume that payment will be successful. It ought to be ready for receiving information from target resource that operation has succeeded or not. As regards software design, active resource is represented by synthesis of three concepts:

- REST resource, which brings unambiguous global identification of resource and convenient language of communication for presentation and change of its state. It also provides availability of many technical frameworks ready for use for implementation of interactions;

- DDD aggregate, which gives a comprehensive way of modeling software external and internal structure, its behavior and rules forming objects identity. DDD technique also brings possibility of effective and cheap implementation thanks to help of modern frameworks for enterprise software such as Spring Framework [16] which was broadly used to implement proposed solution;

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

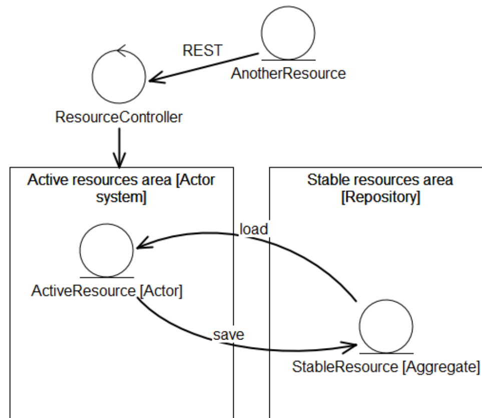

Figure 1: Two areas of resources

- actor, which offers its capability of long-term existence and sophisticated communication abilities. Utilization of actor model is possible and reliable due to existence of mature implementations such as Akka Framework [1], which was used with support of patterns based on it ([2, 18, 20]).

3.2. Two areas of resources

Any resource in the solution may stay in one of two states: stable state, when no change of its properties is possible and active state, when its properties may dynamically change. Therefore, symbolically two areas are distinguished: stable resources area and active resource area as was presented in Fig. 1.

When message to resource in stable area was directed, resource is moved to active area, where it acquires ability to act. If system detects that active resource is idle (does not perform any activity and has empty mailbox), resource may be removed from active area, but it depends on used strategy of supervising. In the system implementation these areas as represented by DDD repository pattern and by actor system respectively. For resource migration into active area, dedicated to such kind of resources, area supervisor is responsible. Sample body of supervisor's function of message handling is shown on Listing 1. Some essential comments have been placed in the code.

Listing 1: Supervisor's message handling function

```
public void onReceive(Object message) {
   if (message instanceof CreateResource) {
      // request to create new resource
      CreateResource msg = (CreateResource) message;
      // create active resource
```

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

```
        ActorRef activeResource =
            context().actorOf(componentNameProps(),
            msg.resourceName());
        // and send message to the newborn in active state.
        // It will be responsible for immediate
        // creation of it's stable representation
        activeResource.tell(msg, self());

    } else if (message instanceof PerformAct) {
        // message to existing resource
        PerformAct msg = (PerformAct) message;
        // is resource in active state?
        ActorRef activeResource =
            this.getContext().getChild(msg.resourceName());
        if (activeResource == null) { // no
            // enter existing resource into active state
            activeResource =
                context().actorOf(componentNameProps(),
                msg.resourceName());
        }
        // send message to resource in active state
        activeResource.tell(msg, self());
    }
}
```

4. Example of distributed batch processing supported by the new concept

Each processing routine in transactional system may be treated as active resource. Examples of such resources are: standing orders processing, interest calculation, interest capitalization, incoming or outgoing payments processing etc. Initialization of processing is therefore implemented as request for creation of resource of some kind. System ordering computation sends request to microservice which is responsible for handling such processing. Typically it will be microservice which owns resources being processed, for example customer contracts or registered payments. It also may be separate microservice as in example below, when processing involves different resources. Ordering system sends POST command with REQUEST performative and in return receives URI of active resource which probably will be created in the target system. Ordering system have to be ready to accept confirmation of resource creation sent by target system – the CONFIRM performative meaning that processing has been started. Thanks to received URI, ordering system is able to contact with active resource in target system and ask it about its state (QUERY_REF performative sent by GET command) or to order further requests, for example hold computation or abandon it (CANCEL performative sent by POST command). Ordering system should be ready to accept information from resource

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

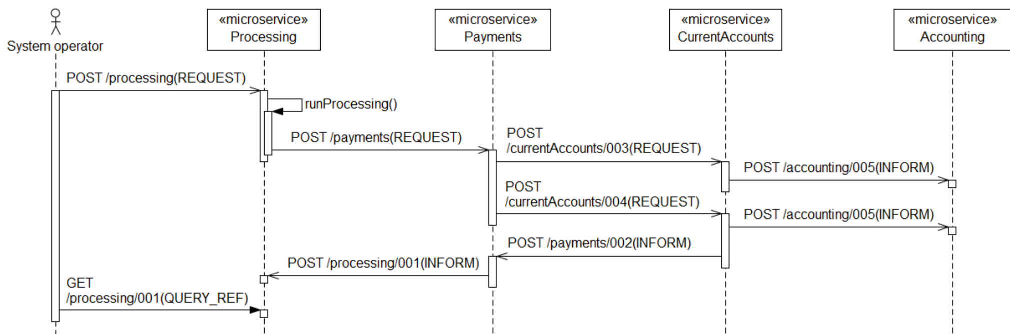

Figure 2: Sample processing supported by the new model

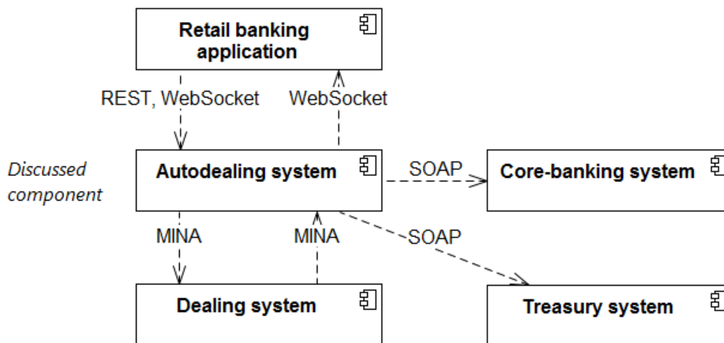

Figure 3: Component model of 'autodealing' example

(which may be sent as INFORM performative by POST command) or explicitly order such information (e.g. results of computation). Sequence diagram presented in Fig. 2 illustrates example of interactions between microservices working on some kind of processing in banking software. For clarity only single set of interactions for one transaction was shown.

5. Example of real-time cooperation and problems solved

Second example concerns real-time computation provided in enterprise system offering auto-dealing functionality, i.e. ability to make unassisted currency exchange transactions on trading platform (typically such transactions are supported by brokers). The whole solution consists of several components. The most important of them are shown in Fig. 3.

The first attempt to implement *Autodealing system* was made using typical service-oriented approach of stateless nature. Implementation and functional tests of the solution

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

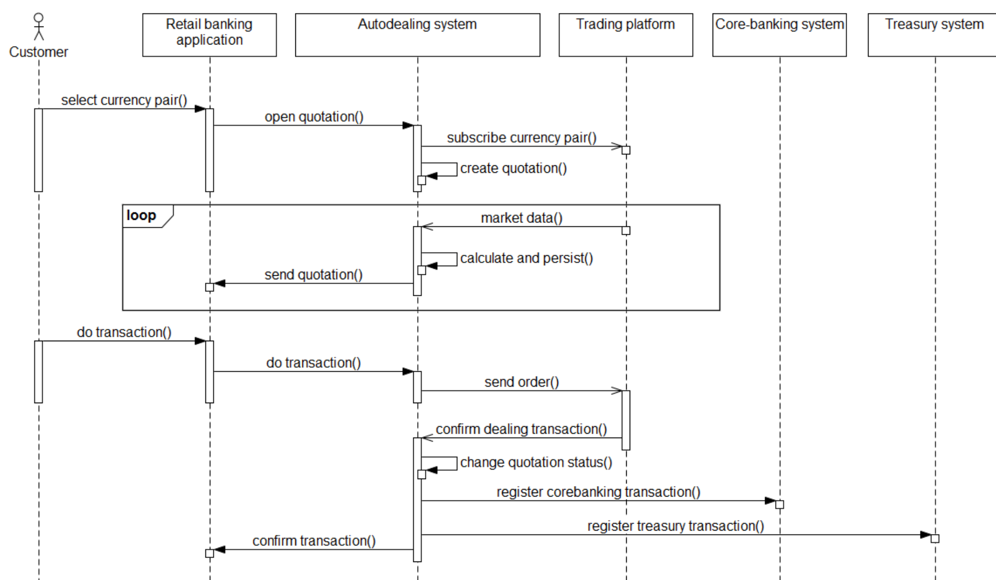

Figure 4: Real-time cooperation supported by service-oriented approach

revealed some problems of diverse origin. In the majority of cases functionally satisfying solution was found, but applied mechanisms were of different type and architecture of the system began to drift into worrying direction. Coherent mechanism was demanded and finally the active resources framework has filled the gap. The most interesting problems and solutions provided by the framework are described below. Description concerns key interactions in system which are shown in Fig. 4.

1. The first problem is connected with events ordering and occurs when first *market data* message arrives when quotation is still being created or persisted and doesn't exist yet. System cannot properly address *market data*. The simplest protection is to ignore *market data* if quotation doesn't exist, but user would wait unacceptably long time if frequency of messages is low (e.g. one per 10 second).

2. The other problem concerns resource access synchronization and appears when frequency of *market data* messages is high and new message arrives when previous is still being handled. System cannot synchronize processing by use of database transactions because remote invocations exist. The simplest way is to ignore incoming messages when the old one is being processed, but it may lead to unwanted business effect of loosing the most beneficial offers. Some advanced queuing mechanism should be used to ensure proper order of servicing.

3. Another problem appears after user decides to finalize transaction and system sends *order* message to *Trading platform*. Until platform doesn't confirm transaction, still *market data* messages arrives to the system. In that state, such messages

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

have to be ignored. Typical solution is to use special *status* property of object and make behavior conditional on its value.

4. One of functional requirements is that user has to be informed about the result of dealing transaction immediately when confirmation from *Trading platform* is got, and the next parts of confirmation handling procedure (interaction with *Core-banking* and *Treasury* domain systems) should be done later. To fulfil such a requirement special tools for asynchronous processing have to be applied.

5. Since communication with domain systems may be unreliable, it should be repeated in controlled way if failure occurs, without blocking main use case scenario. Such a requirement causes that some equipment for background processing have to be used.

6. Until user doesn't select the most interesting currency pair and register private subscription in *Trading platform*, public rates are broadcasted to each observer. To do that, *Autodealing system* have to maintain global list of active observers of each currency pair or list of pairs for each active observer, and therefore special concurrent structure must be applied.

7. Mentioned above list of observers is a static structure, and it cannot be directly used to actively inform observers about communication problems happening, such as temporal unavailability of dealing or domain systems. Low amount of maintenance information causes discomfort and growing impatience when any problem appears.

The active resource framework naturally helps to solve the problems mentioned above.

1. Messages arriving at active resource are enqueued in its mailbox and each of them are processed in exclusive mode (i.e. at the moment resource services only one task). So, hazard cannot occur: creation of resource always would finish before next message is taken to service.

2. All messages in resource mailbox are ordered and are computed exclusively - this property blocks synchronization problem. In both cases (1. and 2.) prioritization function of resource mailbox plays important role: the functional requirement states that when newer message of the same type is already present in queue, it has higher priority and should be served first. Older message should be skipped in such case. Discussed solution supports that requirement.

3. Third problem may be solved by actor capability to dynamically change its behavior. After ordering of transaction, actor switches its handling function to the new one which accepts only confirmation and rejection messages.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

4. Asynchronous processing is the normal way which active resource uses to perform all its tasks. In this example, resource can inform user about *Trading platform* answer and continue cooperation with other systems.

5. Repeated actions may be initiated and controlled by active resource in a few ways. Common pattern is to address message to itself: when error is detected, actor sends message to its own mailbox causing future effect of message processing. Since active resource carries its state, repeated executions may be controlled and, for example, ended after some number of failures.

6. Instead of keeping some kind of global structures storing addresses of observers, built-in function of discussed framework can be used: each type of resource has its own supervisor which keeps mailbox addresses of all its child resources in active state. This address book (which is an actor system function) lets properly broadcast required information to resources, and each resource can inform its front-end client.

7. Each active resource can actively react to problems appearing and can notify connected user in the same way as above. Thereafter it can send information update when conditions are changing.

Generally, one might say, that the new concept more naturally fits requirements of real-time transactional systems then typical service-oriented solution. Examples above point out that it solves many problems of different nature, simplifies software development and introduces coherence into its architecture.

6. DDD tactical tool-set extension

The set of design patterns contained in tactical part of DDD includes the aggregate as a central element and a few additional items. They belong to two sub-layers of business logic layer: *application* and *domain* one. The concept of active resources introduces new sub-layer including additional patterns responsible for active behaviour: the *actors* layer. Enriched model of DDD tactical patterns is shown on Fig. 5.

7. Final remarks

New concept of computation in enterprise software and examples of its application in some processing and real-time routines were presented. Similar solutions work in real banking software and currently is under further research and development, but first results are very promising. After a few first cases of use of the new solution, there might be told that characteristic features of development process based upon new framework are low cost of use cases implementation and low level of defects detected during quality

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

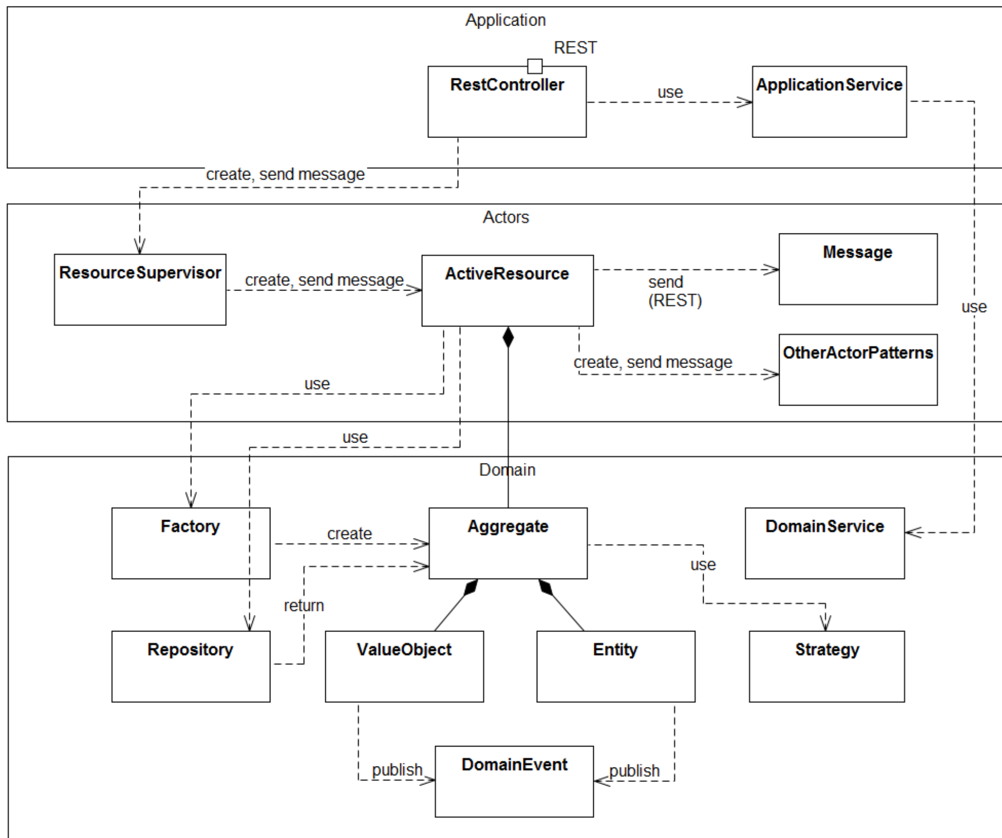

Figure 5: DDD tactical patterns enriched with *actors* layer

assurance phase. The concept of active resource is thought as a broader idea and may serve as fundamental framework able to maintain whole transactional solution. It may be applied to serve both kind of enterprise computation: batch processing, where active resource controls process, and computation of real-time or interactive character, which is more unpredictable and more demanded than batch processing, and active resource helps to control its complexity. Broader research is currently going on. Next planned stage is attempt to build complete new module of transactional system with dominance of resources of the new kind. Additional direction of theoretical and practical development would be step towards utilization of some more sophisticated ideas, such as emergent properties theory, speech acts theory and indeterminism, with hope, that they may help in better understanding of complex software processes.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

References

[1] Akka Framework, http://akka.io. Access 11.2016.

[2] J. ALLEN: Effective Akka. O'Reilly Media, Inc., 2013.

[3] P. BUTCHER: Seven Concurrency Models in Seven Weeks. Pragmatic Bookshelf, 2014.

[4] K.M. CHANDY and L. LAMPORT: Distributed snapshots: Determining global states of Ddistributed systems. *ACM Trans. on Computer Systems*, **3**(1), (1985), 63-75.

[5] E. EVANS: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003.

[6] R.T. FIELDING: Architectural Styles and the Design of Network-based Software Architectures. PhD dissertation, University Of California, Irvine, 2000.

[7] FIPA Standards, http://www.fipa.org/repository. Access 11.2016.

[8] M. FOWLER: Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.

[9] C. HEWITT: Actor Model of Computation: Scalable Robust Information Systems. Cornell University Library, arXiv:1008.1459, 2015.

[10] C. HEWITT, P. BISHOP and R. STEIGER: A universal modular actor formalism for artificial intelligence. *IJCAI'73 Proc. of the 3rd Int. Joint Conf. on Artificial Intelligence*, (1973), 235-245.

[11] S. MILLETT and N. TUNE: Patterns, Principles, and Practices of Domain-Driven Design. John Wiley & Sons, 2015.

[12] S. NEWMAN: Building Microservices. Designing Fine-Grained Systems. O'Reilly Media, 2015.

[13] T. O'CONNOR: Emergent properties. *American Philosophical Quarterly*, **31** (1994), 91-104.

[14] M. NASH and W. WALDRON: Applied Akka Patterns. O'Reilly Media, Inc., 2016.

[15] J.R. SEARLE: Speech Acts. Cambridge University Press, 1969.

[16] Spring Framework, https://spring.io. Access 11.2016.

[17] G. SUKUMAR: Distributed Systems: An Algorithmic Approach. Chapman and Hall/CRC, 2014.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM

[18] V. VERNON: Implementing Domain-Driven Design. Addison-Wesley Professional, 2013.

[19] V. VERNON: Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Addison-Wesley Professional, 2015.

[20] D. WYATT: Akka Concurrency. Artima Press, 2013.

Brought to you by | Politechnika Swietokrzyska - Kielce University of Technology
Authenticated
Download Date | 9/7/17 9:30 AM