Architecture Framework for Mapping Parallel Algorithms to Parallel Computing Platforms

Bedir Tekinerdogan[1], Ethem Arkın[2]

[1]Bilkent University, Dept. of Computer Engineering, Ankara, Turkey
bedir@cs.bilkent.edu.tr
[2]Aselsan MGEO, Ankara, Turkey
earkin@aselsan.com.tr

Abstract. Mapping parallel algorithms to parallel computing platforms requires several activities such as the analysis of the parallel algorithm, the definition of the logical configuration of the platform, and the mapping of the algorithm to the logical configuration platform. Unfortunately, in current parallel computing approaches there does not seem to be precise modeling approaches for supporting the mapping process. The lack of a clear and precise modeling approach for parallel computing impedes the communication and analysis of the decisions for supporting the mapping of parallel algorithms to parallel computing platforms. In this paper we present an architecture framework for modeling the various views that are related to the mapping process. An architectural framework organizes and structures the proposed architectural viewpoints. We propose five coherent set of viewpoints for supporting the mapping of parallel algorithms to parallel computing platforms. We illustrate the architecture framework for the mapping of array increment algorithm to the parallel computing platform.

Keywords: Model Driven Software Development, Parallel Programming, High Performance Computing, Domain Specific Language, Modelling.

1 Introduction

It is now increasingly acknowledged that the processing power of a single processor has reached the physical limitations and likewise serial computing has reached its limits. To increase the performance of computing approaches the current trend is towards applying parallel computing on multiple nodes typically including many CPUs. In contrast to serial computing in which instructions are executed serially, in parallel computing multiple processing elements are used to execute the program instructions simultaneously.

One of the important challenges in parallel computing is the mapping of the parallel algorithm to the parallel computing platform. The mapping process requires several activities such as the analysis of the parallel algorithm, the definition of the logical configuration of the platform, and the mapping of the algorithm to the logical configuration platform. Based on the analysis of the algorithm several design decisions for allocating the algorithm sections to the logical configurations must be made. To support the communication among the stakeholders, to reason about the design decisions during the mapping process and to analyze the eventual design it is important to adopt the appropriate modeling approaches. In current parallel computing approaches there does not seem to be standard modeling approaches for supporting the mapping process. Most approaches seem to adopt conceptu-

al modeling approaches in which the parallel computing elements are represented using idiosyncratic models. Other approaches borrow for example models from embedded and real time systems and try to adapt these for parallel computing. The lack of a clear and precise modeling approach for parallel computing impedes the communication and analysis of the decisions for supporting the mapping of parallel algorithms to parallel computing platforms.

In this paper we present an architecture framework for modeling the various views that are related to the mapping process. An architectural framework organizes and structures the proposed architectural viewpoints. We propose five coherent set of viewpoints for supporting the mapping of parallel algorithms to parallel computing platforms. We illustrate the architecture framework for the mapping of parallel array increment algorithm to the parallel computing platform.

The remainder of the paper is organized as follows. In section 2, we describe the background on software architecture viewpoints and define the parallel computing metamodel. Section 3 presents the viewpoints based on the defined metamodel. Section 4 presents the guidelines for using the viewpoints. Section 5 presents the related work and finally we conclude the paper in section 6.

2 Background

In section 2.1 we provide a short background on architecture viewpoints which is necessary for defining and understanding the viewpoint approach. Subsequently, in section 2.2 we provide the metamodel for parallel computing that we will later use to define the architecture viewpoints in section 3.

2.1 Software Architecture Viewpoints

To represent the mapping of parallel algorithm to parallel computing platform it is important to provide appropriate modeling approaches. For this we adopt the modeling approaches as defined in the software architecture design community. According to ISO/IEC 42010 the notion of *system* can be defined as a set of components that accomplishes a specific function or set of functions[4]. Each system has an architecture, which is defined as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution". A common practice to model an architecture of a software intensive system is to adopt different architectural views for describing the architecture according to the stakeholders' concerns [2]. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. An architectural viewpoint defines the conventions for constructing, interpreting and analyzing views. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An *architectural framework* organizes and structures the proposed architectural viewpoints [4]. The concept of *architectural view* appears to be at the same level of the concept of *model* in the model-driven development approach. The concept of *viewpoint*, representing the language for expressing views, appears to be on the level of metamodel. From the model-driven development perspective, an architecture framework as such can be considered as a coherent set of domain specific languages [9]. The notion of architecture framework and the viewpoints plays an important role in modeling and documenting architectures. However, the existing approaches on architecture modeling seem to

have primarily focused on the domain of traditional, desktop-based and sometimes distributed development platforms. Parallel computing systems have not been frequently or explicitly addressed.

2.2 Parallel Computing Metamodel

Fig. 1 shows the abstract syntax of the metamodel for mapping parallel algorithms to parallel computing platform. The metamodel consists of four parts including *Parallel Algorithm, Physical Configuration, Logical Configuration,* and *Code.* In the *Parallel Algorithm* part we can observe that an *Algorithm* consists of multiple *Sections,* which can be either *Serial Section* or *Parallel Section.* Each section is mapped on *Operation* which on its turn is mapped on *Tile.*

Physical Configuration represents the physical configuration of the parallel computing platform and consists of *Network* and *Nodes. Network* defines the communication medium among the *Nodes. Node* consists of *Processing Unit* and *Memory*. Since a node can consist of multiple processing units and memory units we assume that different configurations can be defined including shared memory and distributed memory architectures. *Logical Configuration* represents a model of the physical configuration that defines the logical communication structure among the physical nodes. *LogicalConfiguration* consists of a number of *Tiles*. *Tile* can be either a (single) *Core*, or *Pattern* that represents a composition of tiles. Patterns are shaped by the operations of the sections in the algorithm. *Pattern* includes also the communication links among the cores. The algorithm sections are mapped to *CodeBlocks.* Hereby, *SerialSection* is implemented as *SerialCode*, and *ParallelSection* as *ParallelCode.* Besides of the characteristic of *ParallelSection* the implementation of *ParallelCode* is also defined by *Pattern* as defined in the logical configuration. The overall *Algorithm* is run on *PhysicalConfiguration.*

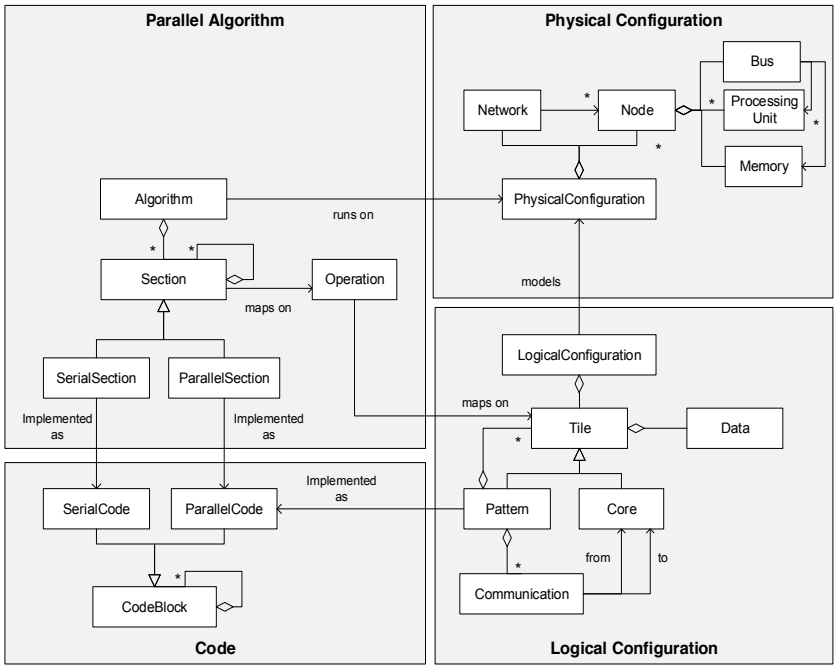


Fig. 1. Metamodel for Mapping Parallel Algorithm to Parallel Computing Platform

3 Architecture Viewpoints for Parallel Computing

Based on the metamodel of Fig. 1 we define the architecture framework consisting of a coherent set of viewpoints for supporting mapping of parallel algorithms to parallel computing platform. In section 3.1 we will first describe an example parallel algorithm and the corresponding logical configuration. In section 3.2 we will present the *algorithm decomposition viewpoint*. In section 3.3 we will present *the physical configuration viewpoint*. Section 3.4 presents the logical configuration viewpoint, section 3.5 the algorithm-to-logical configuration viewpoint, and finally, section 3.6 will present the algorithm-to-code viewpoint.

3.1 Case Description

To illustrate the problem we will use the array increment algorithm as shown in Fig. 2 that will be mapped on a 4x4 physical parallel computing architecture. Given an array the algorithm recursively decomposes the array into sub-arrays to increment each element with one. The algorithm is actually composed of two different parts. In the first part the array element is incremented with one if the array size is one (line 3). If the array size is greater than one then in the second part the array is decomposed into two sub-arrays and the algorithm is recursively called.

```
1.  Procedure ArrayInc(A[], n):
2.  if n=1 then
3.    *A += 1
4.  else
5.    ArrayInc(A, n/2)
6.    ArrayInc(A+n/2, n)
7.  endif
```

Fig. 2. Array Increment Algorithm

3.2 Algorithm Decomposition Viewpoint

In fact the array increment algorithm is a serial (recursive) algorithm. To increase the time performance of the algorithm we can map it to a parallel computing platform and run it in parallel. For this it is necessary to decompose the algorithm into separate sections and define which sections are serial and which can be run in parallel. Further, each section of an algorithm realizes an operation, which is a reusable abstraction of a set of instructions. For serial sections the operation can be custom to the algorithm. For parallel sections in general we can identify for example the primitive operations *Scatter* for distributing data to other nodes, *Gather* for collecting data from nodes, *Broadcast* for broadcasting data to other nodes, etc. Table 1 shows the algorithm decomposition viewpoint that is used to decompose and analyze the parallel algorithm. The viewpoint is based on the concepts of the *Parallel Algorithm* part of the metamodel in Fig. 1. An example algorithm decomposition view that is based on this viewpoint is shown in Fig. 3A. Here we can see that the array increment algorithm has been decomposed into four different sections with two serial and two parallel sections. Further, for each section we have defined its corresponding operation.

3.3 Physical Configuration Viewpoint

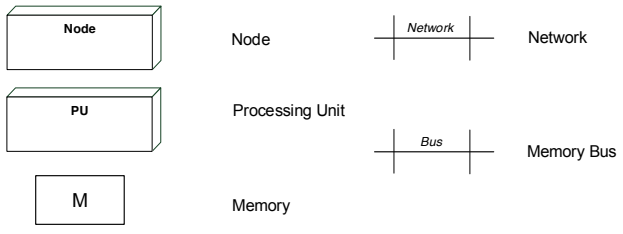
Table 2 shows the physical configuration viewpoint for modeling the parallel computing architecture. The viewpoint is based on the concepts of the *Physical Configuration* part of

the metamodel in Fig. 1. As we can see from the table the viewpoint defines explicit notations for *Node, Processing Unit, Network, Memory Bus* and *Memory.* An example physical configuration view that is based on this viewpoint is shown in Fig. 3B. Here the physical configuration consists of four nodes interconnected through a network. Each node has four processing units with a shared memory. Both the nodes and processing units are numbered for identification purposes.

Table 1. Algorithm Decomposition Viewpoint

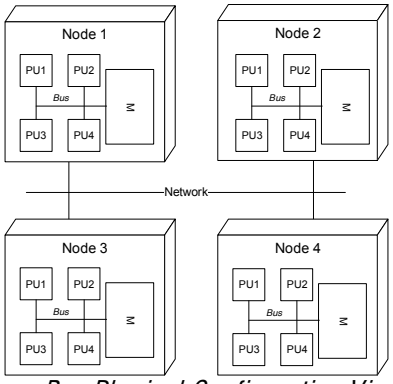
Name	Algorithm Decomposition Viewpoint
Concerns	Decomposing an algorithm into different sections which can be either serial or parallel. Analysis of the algorithm.
Stakeholders	Algorithm analysts, logical configuration architect, physical configuration architect
Elements	• Algorithm – represents the parallel algorithm consisting of sections. • Serial Section – a part of an algorithm consisting of a coherent set of instructions that needs to run in serial • Parallel Section – a part of an algorithm consisting of a coherent set of instructions that needs to run in parallel • Operation – abstract representation of the set of instructions that are defined in the section
Relations	• Decomposition relation defines the algorithm and the sections
Constraints	• A section can be either SER or PAR, not both
Notation	*Index* | *Algorithm Section* | *Section Type* | *Operation*

Table 2. Physical Configuration Viewpoint

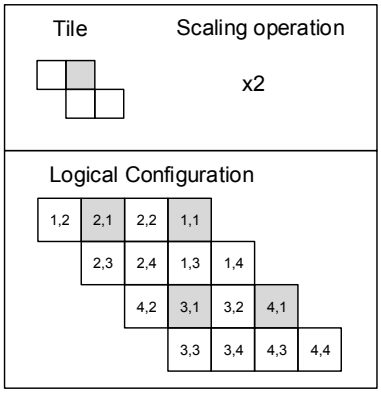
Name	Physical Configuration Viewpoint
Concerns	Defining physical configuration of the parallel computing platform
Stakeholders	Physical configuration architect
Elements	• Node – A standalone computer usually comprised of multiple CPUs/ /cores, memory, network interfaces, etc. • Network – medium for connecting nodes • Memory Bus – medium for connecting processing units within a node • Processing Unit – processing unit that reads and executes program instructions • Memory – unit for data storage
Relations	• Nodes are networked together to comprise a supercomputer • Processing units are connected through a bus
Constraints	• Processing Units can be allocated to Nodes only • Memory can be shared or distributed
Notation	

Ind.	Algorithm Section	Section Type	Operation
1	A1 = [A, A+n/2] A2 = [A+n/2, A+n]	SER	*Decompose*
2	Distribute the sub-arrays	PAR	*Scatter*
3	*A += 1	SER	*Increment*
4	Collect increment-ed array results	PAR	*Gather*

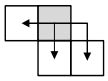
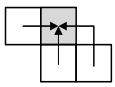
A. Algorithm Decomposition View



B. Physical Configuration View



C. Logical Configuration View

Ind.	Algorithm Section	Plan
1	A1=[A, A+n/2] A2=[A+n/2, A+n]	Run on each node
2	Distribute the sub-arrays	
3	*A += 1	Run on each node
4	Collect incre-mented array results	

D. Algorithm-to-Logical Configuration View

NO	Algorithm Section	Code Block
1	A1=[A, A+n/2] A2=[A+n/2, A+n]	A1 = [A, A+n/4] A2 = [A+n/4, A+n/2] A3 = [A+n/2, A+3n/4] A4 = [A+3n/4, A+n]
2	Distribute the sub-arrays	Will be generated
3	*A += 1	*A += 1
4	Collect incremented array results	Will be generated

E. Algorithm-to-Code View

Fig. 3. Views for the given case using the defined viewpoints

3.4 Logical Configuration Viewpoint

Table 3 shows the logical configuration viewpoint for modeling the logical configuration of the parallel computing architecture. The viewpoint is based on the concepts of the *Logical Configuration* part of the metamodel in Fig. 1. The viewpoint defines explicit notations

for *Core, Dominating Core, Tile, Pattern* and *Communication.* An example logical configuration view that is based on this viewpoint is shown in Fig. 3C. The logical configuration is based on physical configuration as shown in Fig. 3B and shaped according to the algorithm in Fig. 3A. As we can observe from the figure the 16 cores in the four nodes of the physical configuration are now rearranged to implement the algorithm properly. Each core is numbered based on both the node number and core number in the physical configuration. For example, core (2,1) refers to the processing unit 1 of physical node 2. Typically, for the same physical configuration we can have many different logical configurations each of them indicating different communication and exchange patterns of data among the cores in the nodes. In our earlier paper we define an approach for deriving the feasible logical configurations with respect to speed-up and efficiency metrics [1]. In this paper we assume that a feasible logical configuration is selected. For very large configurations such as in exascale computing [5] it is not feasible to draw this on the same scale. Instead we define the configuration as consisting of a set of tiles which are used to generate the actual logical configuration. In the example of Fig. 3C we can see that the configuration can be defined as a tile that is two times recursively scaled to generate the logical configuration. For more details about the scaling process we refer to our earlier paper [1].

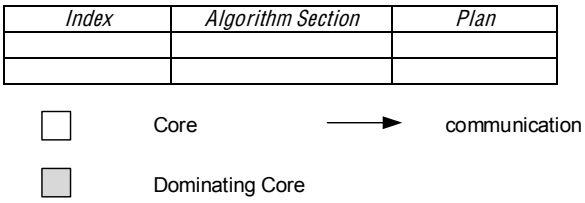
Table 3. Logical Configuration Viewpoint

Name	Logical Configuration Viewpoint
Concerns	Modeling of the logical configuration for the physical configuration
Stakeholders	Logical configuration architect
Elements	• Core – model of processing unit • Dominating Core – the processing unit that is responsible for exchanging data with other nodes
Relations	• Cores can be composed into larger tiles • Tiles can be used to define/generate logical configuration
Constraints	• The number of cores should be equal to the processing units in the physical configuration • The numbering of the cores should match the numbering in the physical configuration
Notation	n,p Core — n - the id of the node in the physical configuration; p - the id of the processing unit in the physical configuration n,p Dominating Core

3.5 Algorithm-to-Logical Configuration Mapping Viewpoint

The logical configuration view represents the static configuration of the nodes to realize the parallel algorithm. However, it does not illustrate the communication patterns among the nodes to represent the dynamic behavior of the algorithm. For this the algorithm-to-logical configuration mapping viewpoint is used. The viewpoint is illustrated in Table 4. For each section we describe a plan that defines on which nodes the corresponding operation of the section will run. For serial sections usually this is a custom operation. For parallel section the plan includes the communication pattern among the nodes. A communication pattern includes communication paths that consist of a source node, a target node and a route between the source and target nodes. An algorithm-to-logical configuration mapping view is represented using a table as it is, for example, shown in Fig. 3D.

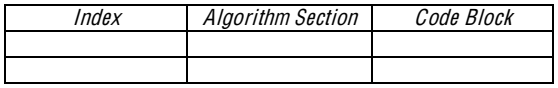
Table 4. Algorithm-to-Logical Configuration Mapping Viewpoint

Name	Algorithm-to-Logical Configuration Mapping Viewpoint
Concerns	Mapping the communication patterns of the algorithm to the logical configuration
Stakeholders	System Engineers, Logical configuration architect
Elements	• Section – a part of an algorithm consisting of a coherent set of instructions. A section is either serial or parallel • Plan – the plan for each section to map the operations to the logical configuration units • Core – model of processing unit • Dominating Core – the processing unit that is responsible for exchanging data with other nodes • Communication – the communication pattern among the different cores
Relations	• Mapping of plan to section
Constraints	• Each serial section has a plan that defines the nodes on which it will run • Each parallel section has a plan that defines the communication patterns among nodes
Notation	

3.6 Algorithm-to-Code Viewpoint

Once the logical configuration and the corresponding algorithm section allocation plan has been defined, the implementation of the algorithm can be started. The corresponding viewpoint is shown in Table 5. The viewpoint is based on the concepts of the *Parallel Algorithm* and *Code* parts of the metamodel in Fig. 1. An example algorithm decomposition view that is based on this viewpoint is shown in Fig. 3E.

Table 5. Algorithm-to-Code Mapping Viewpoint

Name	Algorithm-to-Code Mapping
Concerns	Mapping the algorithm sections to code
Stakeholders	Parallel Programmer, System Engineer
Elements	• Algorithm – represents the parallel algorithm consisting of sections. • Section – a part of an algorithm consisting of a coherent set of instructions • Code Block – code for implementing the section
Relations	• Realization of the section to code
Constraints	• Each section has a code block
Notation	

4 Guidelines for Adopting Viewpoints

In the previous section we have provided the architecture framework consisting of a coherent set of viewpoints for supporting the mapping of parallel algorithms to parallel com-

puting platforms. An important issue here is of course the validity of the viewpoints. In general evaluating architecture viewpoints can be carried out from various perspectives including the appropriateness for stakeholders, the consistency among viewpoints, and the fitness of the language. We have evaluated the architecture framework according the approach that we have described in our earlier study [9]. Fig. 4 shows the process as a UML activity for adopting the five different views. The process starts initially with the definition of algorithm to decomposition view and the physical configuration view, which can be carried out in parallel. After the physical configuration view is defined the logical configuration view can be defined, followed by the modeling of the algorithm to logical view, and finally the algorithm to code view. Among the different steps several iterations can be required which is shown by the arrows.

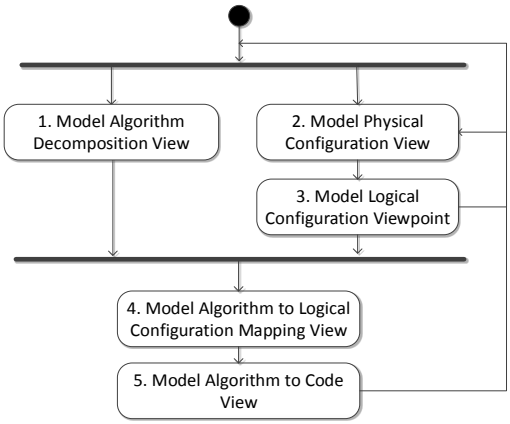


Fig. 4. Approach for Generating/Developing and Deployment of Parallel Algorithm Code

5 Related Work

In the literature of parallel computing the particular focus seems to have been on parallel programming models such as MPI, OpenMP, CILK etc. [8] but the design and the modeling got less attention. Several papers have focused in particular on higher level design abstractions in parallel computing and the adoption of model-driven development.

Several approaches have been provided to apply model-driven development to high performance computing. Similar to our approach Palyart et. al. [7] propose an approach for using model-driven engineering in high performance computing. They focus on automated support for the design of a high performance computing application based on abstract platform independent model. The approach includes the steps for successive model transformations that enrich progressively the model with platform information. The approach is supported by a tool called Archi-MDE. Gamatie et al. [3] represent the Graphical Array Specification for Parallel and Distributed Computing (GASPARD) framework for massively parallel embedded systems to support the optimization of the usage of hardware resources. GASPARD uses MARTE standard profile for modeling embedded systems at a high abstraction level. MARTE models are then refined and used to automatically generate code. Our approach can be considered an alternative approach to both GASPARD and Archi-MDE. The difference of our approach is the particular focus on optimization at the design level using architecture viewpoints.

Several hardware/software codesign approaches for embedded systems start from high level designs from which the system implementation is produced after some automatic or manual refinements. However, to the best of our knowledge no architecture viewpoints have been provided before for supporting the parallel computing engineer in mapping the parallel algorithm to parallel computing platform.

6 Conclusion

In this paper we have provided an architecture framework for supporting the mapping of parallel algorithms to parallel computing platforms. For this we have first defined the metamodel that includes the underlying concepts for defining the viewpoint. We have evaluated the viewpoints from various perspectives and illustrated it for the array increment algorithm. We were able to apply the viewpoint for the incrementing array algorithm. We have adopted the approach also for other parallel algorithms without any problem. Adopting the viewpoints enable the communication among the parallel computing stakeholders, the analysis of the design decisions and the implementation of the parallel computing algorithm. In our future work we will define the tool support for implementing the viewpoints and we will focus on depicting the design space of configuration alternatives and the selection of feasible alternatives with respect to the relevant high performance computing metrics.

References

1. Arkin, E., Tekinerdogan, B., Imre, K. 2013. Model-Driven Approach for Supporting the Mapping of Parallel Algorithms to Parallel Computing Platforms. *Proc. of the ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems.*
2. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.
3. Gamatié, A., et. al. 2011. A Model-Driven Design Framework for Massively Parallel Embedded Systems. *ACM Transactions on Embedded Computing Systems*, *10*(4), 1–36, 2011.
4. ISO/IEC 42010:2007] Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010), 2011.
5. Kogge, P. et al., *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*. DARPA. (2008)
6. Object Management Group (OMG). http://omg.org, accessed: 2013.
7. M. Palyart, D.Lugato, I.Ober, and J.M. Bruel. MDE4HPC: an approach for using model-driven engineering in high-performance computing. In Proc. of the 15th Int. Conf. on Integrating System and Software Modeling (SDL'11), Iulian Ober and Ileana Ober (Eds.). Springer-Verlag, 2011.
8. D. Talia. 2001. Models and Trends in Parallel Programming. *Parallel Algorithms and Applications* 16, no. 2: 145-180.
9. B. Tekinerdogan, E. Demirli. Evaluation Framework for Software Architecture Viewpoint Languages. in Proc. of Ninth International ACM Sigsoft Conference on the Quality of Software Architectures Conference (QoSA 2013), Vancouver, Canada, pp. 89-98, June 17-21, 2013.