MODEL-DRIVEN DESIGN FOR MAPPING PARALLEL APPLICATIONS TO PARALLEL COMPUTING PLATFORMS

1st International Workshop on Advanced Topics in Software Engineering
Theme: Model-Driven Software Development

November 7, 2014
Istanbul, Turkey

Bedir Tekinerdogan

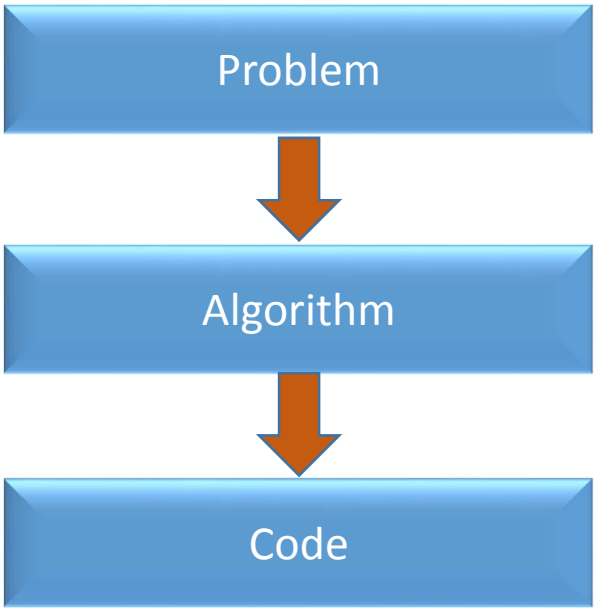
Dept. of Computer Engineering
Bilkent University, Turkey

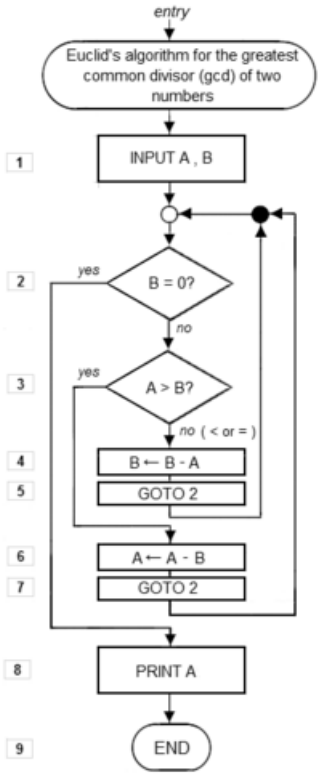
Possible Presentation Topics

- Model-Driven Development for Software Architecture Design
- Model-Driven Software Product Line Engineering
- Model-Driven Development for Global Software Development
- Model-Based UI Design
- Model-Driven Development and Aspect-Oriented Software Development
- Model-Based Testing for Safety-Critical Systems

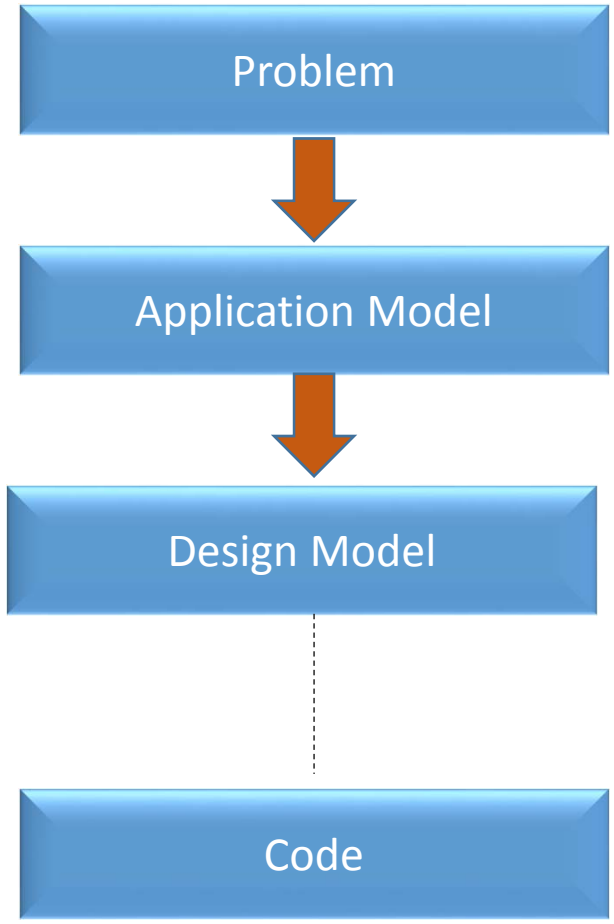
- Model-Driven Design for Parallel Computing

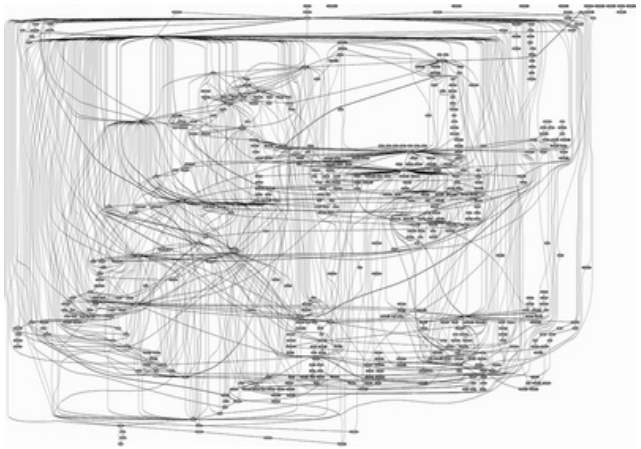
The Reductionist View – Algorithm Design and Analysis



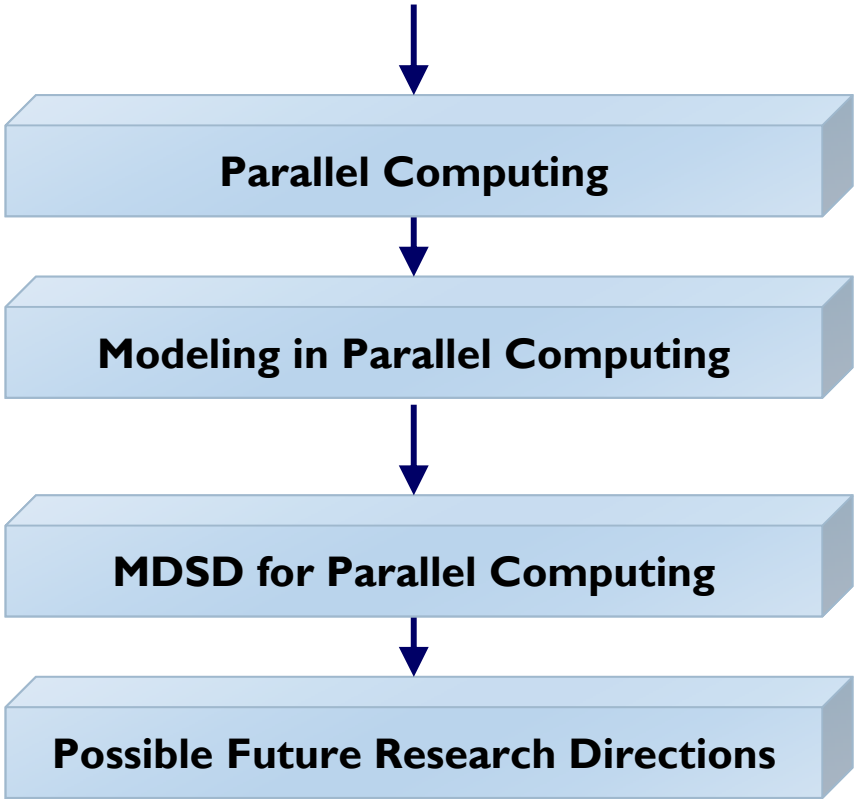


The Holistic View – Software Engineering ☺





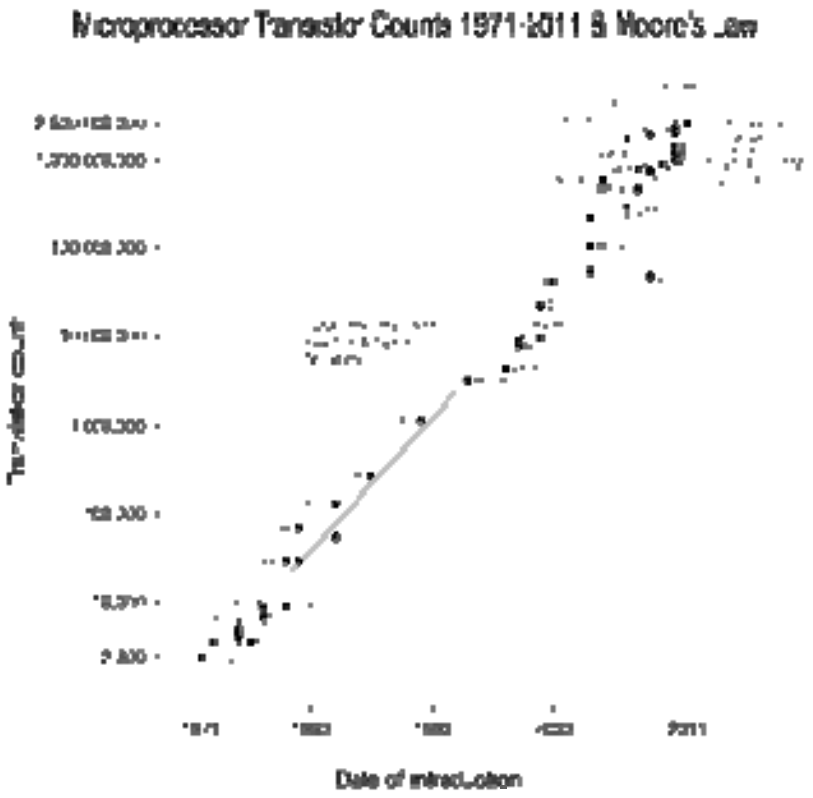
Contents


Parallel Computing
Modeling in Parallel Computing
MDSD for Parallel Computing
Possible Future Research Directions

Parallel Computing

Moore's Law…

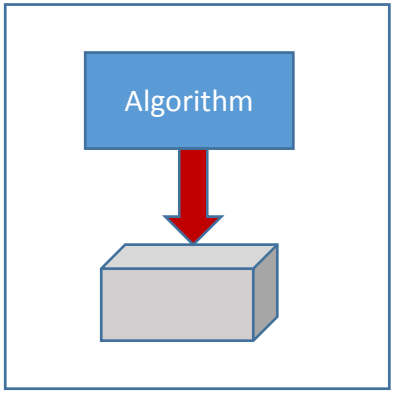
- The famous Moore's law states that the number of transistors on integrated circuits and likewise the **performance of processors doubles approximately every eighteen months**.
- Although Moore's law is still in effect, currently it is recognized that increasing **the processing power of a single processor has reached the physical limitations**
- between 1986 and 2002
  - performance to improve by 52% per yearSince 2002,
  - Performance has improved less than 20% per year.



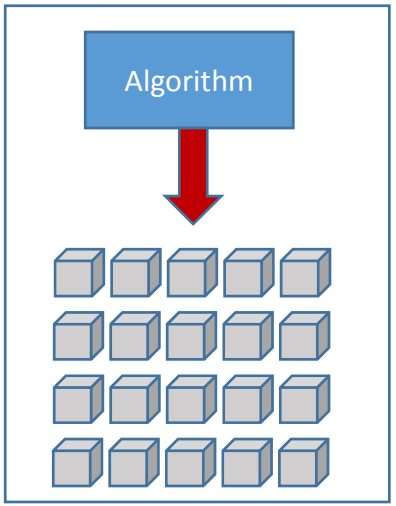
http://en.wikipedia.org/wiki/Moore's_law

Need for Parallel Computing

- To increase the performance the **current trend is towards applying parallel computing on multiple nodes**.



- Here, unlike serial computing in which instructions are executed serially, **multiple processing elements** are used to execute the program instructions simultaneously



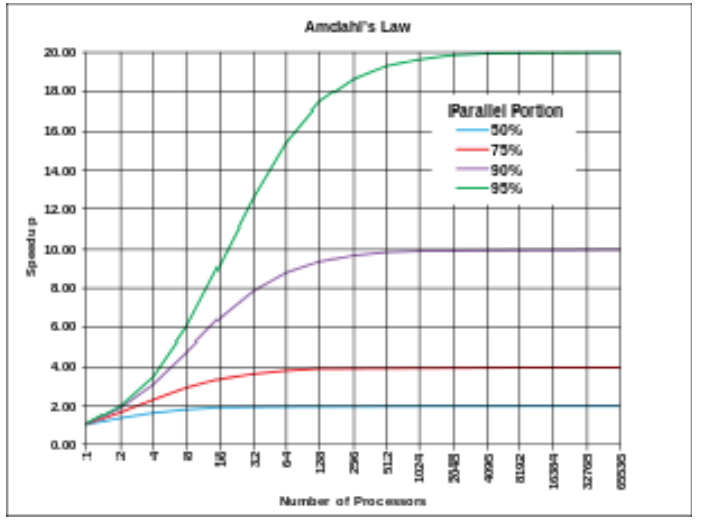
Parallel Algorithm

- To benefit from the parallel computing power, **parallel algorithms are defined** to be executed simultaneously on multiple nodes.

- The adoption of parallel algorithms executed on parallel computing platforms will increase the performance of the execution of the programs wrt serial computing (*speedup*)

- Ahmdal's Law
  states that potential program speedup is defined by the fraction of code (P) that can be parallelized:

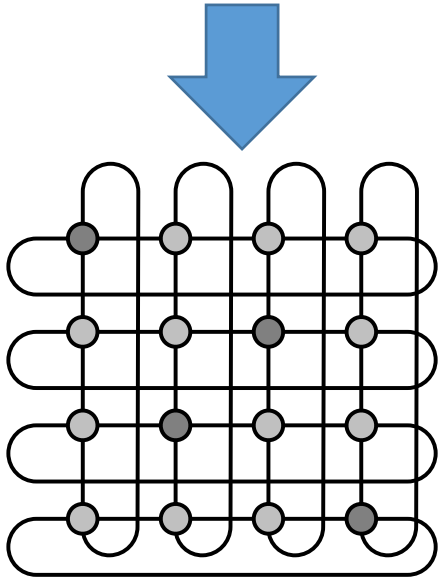
$$speedup = \frac{1}{1 - P}$$



http://en.wikipedia.org/wiki/Amdahl's_law

Mapping Parallel Algorithm to Physical Computing Platform

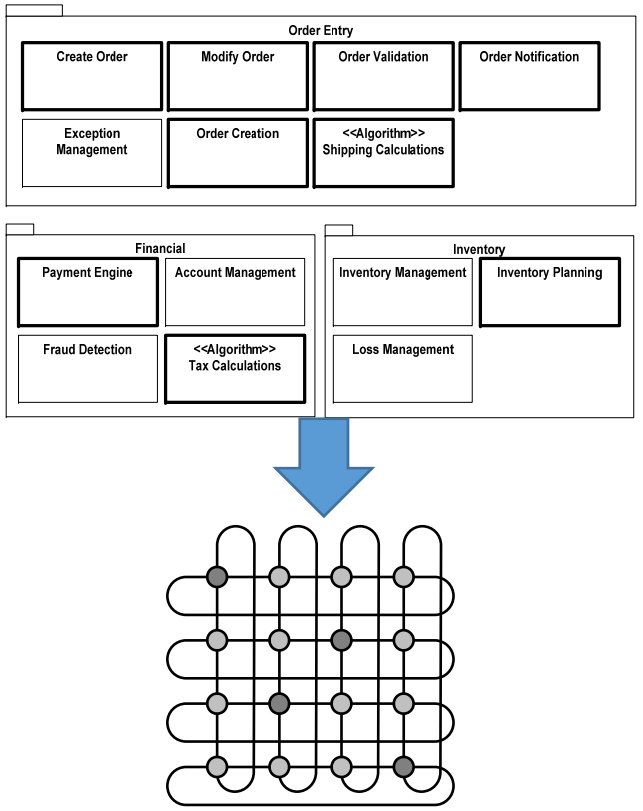
```
Procedure Complete-Exchange:
for i=1 to n-1
   Collect data to the dominating nodes from the dominated nodes
Endfor
for i=n-1 downto 1
   Exchange the selected data between dominating nodes
   Distribute data from dominating nodes to the dominated nodes
Endfor
```



Parallel Computing Platform (Physical Configuration)

Beyond Algorithms…
Mapping Parallel Applications to Parallel Platforms

- A close analysis of parallel computing research shows that the well-defined concept of *algorithm* is prevailing and
- the broader consideration of software *application* and its mapping to parallel computing platform does not seem to have got much attention.



Parallel Computing Platform
(Physical Configuration)

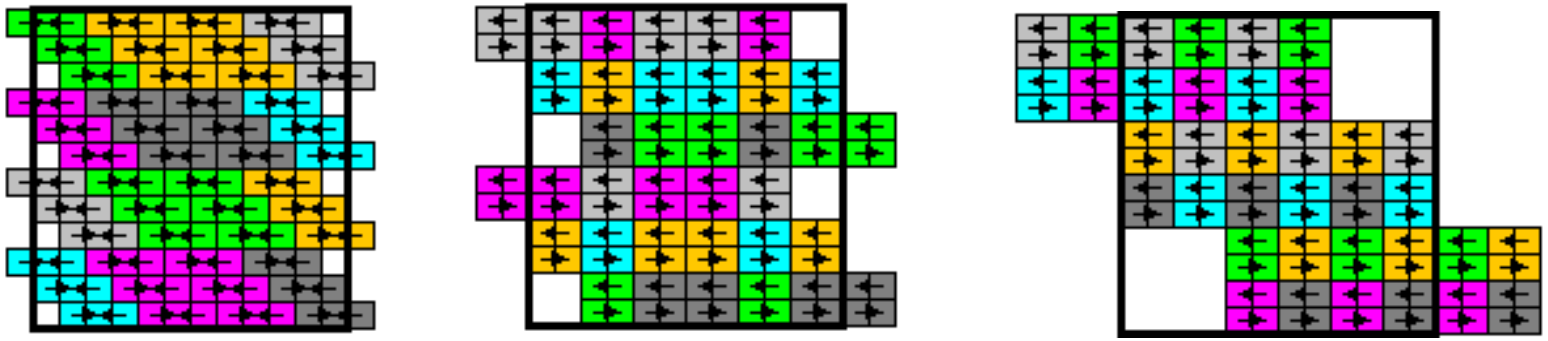
Mapping Parallel Algorithm/Application to Logical Configuration Platform

The *logical configuration* is a view of the physical configuration that defines the logical communication structure among the physical nodes.

```
Procedure Complete-Exchange:
for i=1 to n-1
   Collect data to the dominating nodes from the dominated nodes
Endfor
for i=n-1 downto 1
   Exchange the selected data between dominating nodes
   Distribute data from dominating nodes to the dominated nodes
Endfor
```

Possible Logical Configurations

Many alternative configurations



How to select the feasible configuration?

- **Speedup** – defines relative performance improvement when executing a task

$$S_p = \frac{T_s}{T_p}$$

where Ts is the execution time of the serial algorithm.
Tp is the execution time of the parallel algorithm with p processors

- **Efficiency** – defines how well the processors are utilized in executing the algorithm

$$E_p = \frac{S_p}{p}$$

where Sp is the speed up as defined in the above equation
and p is the number of processors

Problem Statement

- For a given parallel algorithm/application and physical configuration we need to **define and generate the possible logical configurations** and accordingly the mapping alternatives.
- Once a feasible mapping alternative is selected **the required code for the parallel computing platform** needs to be provided to realize the parallel algorithm.
- Due to the complexity and size of the mapping problem **it is not easy to implement the parallel algorithm/application manually** on the parallel computing platforms.
- Obviously, a **systematic approach that is supported by tools is necessary** to analyze the parallel algorithm/application, model the logical configuration, select feasible mapping alternatives and generate the code for the computing platform.

Modeling in Parallel Computing

Parallel Computing- Observation

Lack of Precise Models

- In current parallel computing approaches **standard modeling approaches for supporting the mapping process seems to be lacking**.
- Most approaches seem to adopt more **informal conceptual 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.

Manual Process

- The mapping process is **largely a manual process** and needs better automation support

Modeling the Architectural Deployment

The deployment should be explicitly and carefully modeled for supporting:

- **Communication among stakeholders**
  - To reason about design decisions related to the mapping of parallel algorithms to parallel computing platforms.
- **Analysis**
  - to support the analysis of selected configurations and the design decisions related to the mapping including speedup and efficiency
- **Support development**
  - to support the detailed design & implementation based on the provided configuration

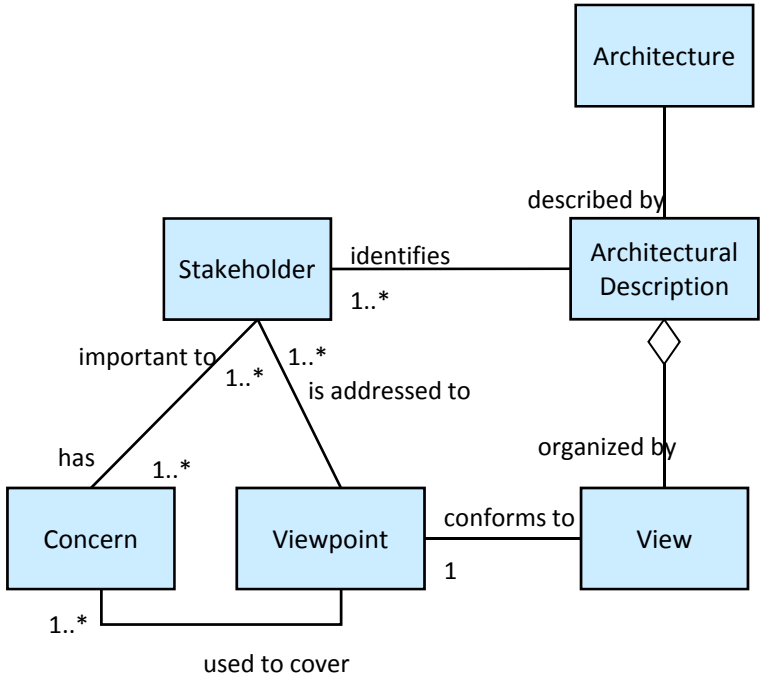
Architectural Viewpoints

View:

- a representation of a system from the perspective of one or more concerns which are held by one or more stakeholders.

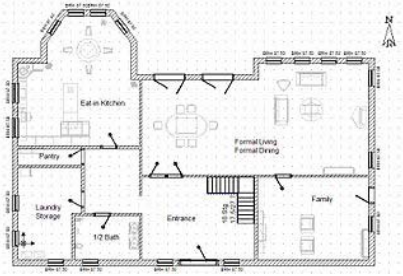
Viewpoint:

- A pattern or template from which to construct individual views.

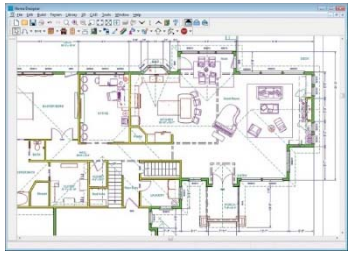


[ISO/IEC 42010:2007] Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010) July 2007.

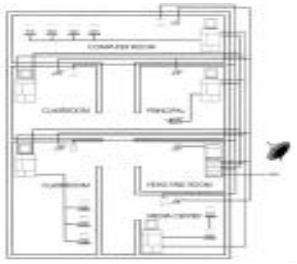
Multiple Views of the Architecture



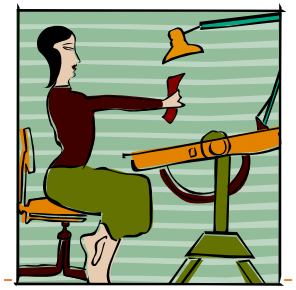
Floor plan



Interior Plan



Wiring Plan



Architecture Viewpoint Approaches

Different viewpoint approaches have been introduced which include viewpoints for mapping software elements to nodes

Kruchten's 4+1 viewpoint approach
- Logical View
- Process View
- Development View
- Physical View
- Scenario view

UML 4+1 Viewpoint approach
- Design View
- Process View
- Implementation View
- Deployment View
- Use case view

Views and Beyond approach (V&B)
- Module Viewtype
- C&C Viewtype
- Allocation Viewtype

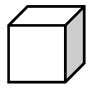
Siemens Viewpoint approach
- Conceptual view,
- Module interconnection view,
- Execution view,
- Code view.

Etc.

Example – UML Deployment Viewpoint

Viewpoint

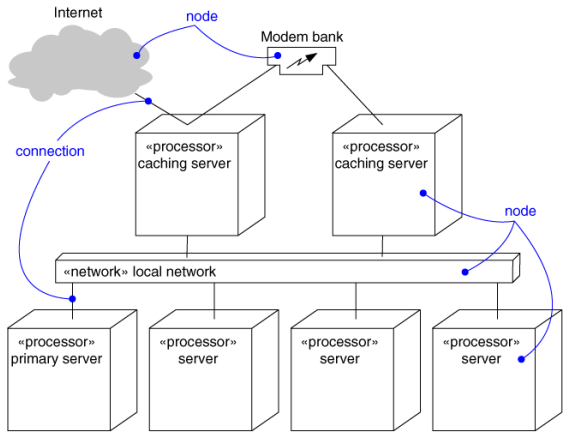
- Name: Deployment Viewpoint
- Stakeholders:
  - System Designer
- Concerns:
  - System Design
- Components:
  - Processing Nodes
- Notation



Node

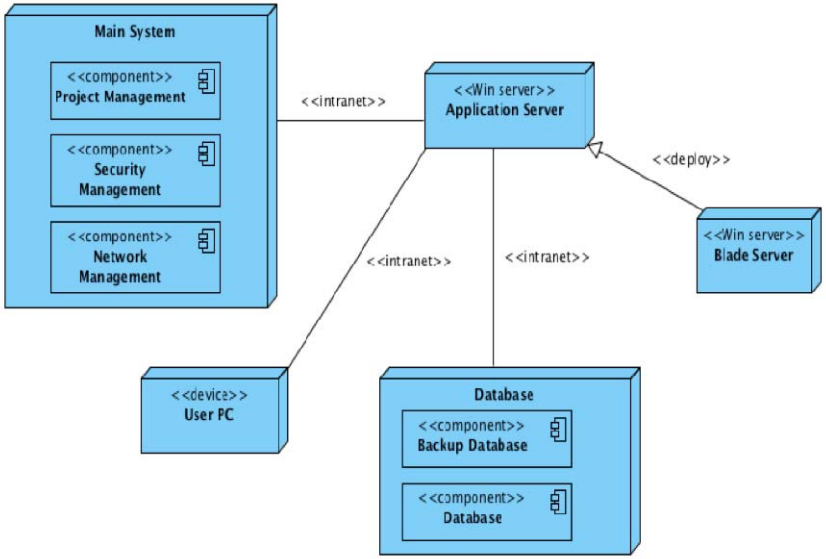
Connection

Deployment View - Example



Deployment Viewpoint (Style) V&B Approach

- **Element**:
  - Software element: usually elements from the C&C viewtype.
  - Environmental element: computing hardware – processor, memory, disk, network, and so on.



- **Relations**:
  - *Allocated-to,* showing on which physical elements the sofware resides
  - *Migrates-to,* if the allocation is dynamic.

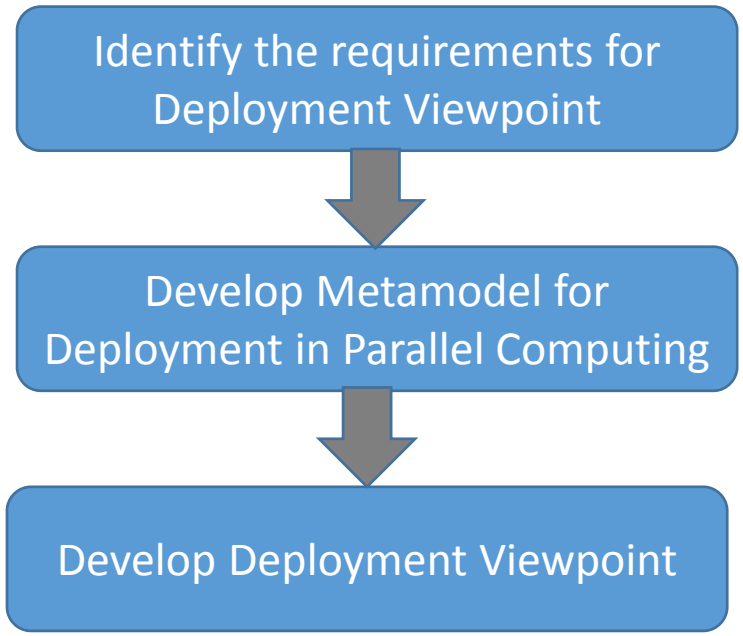
- **Topology**
  - The allocation topology is unrestricted. However, the required properties of the software must be satisfied by the provided properties of the hardware.

Deployment Viewpoints for Parallel Computing

- To represent the different concerns of deployments in parallel computing we could consider adopting the existing general-purpose viewpoints.

- Unfortunately, these general-purpose frameworks fall short for expressing the particular concerns for parallel computing
  - the derived deployment views are usually **visual notations** that are basically targeted for human designers and as such the deployment needs to be done manually.
  - visual models are suitable for small to medium applications but soon they **do not scale well** for deployment of large scale parallel applications.

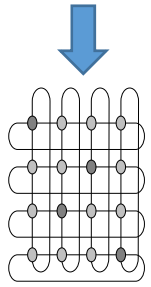
Approach for Developing Deployment Viewpoint for Mapping Process



Requirements for Deployment Viewpoint for Mapping Parallel Algorithms/Applications

- ***Modeling of the physical computing platform***
  The DSL needs to be able to model the physical computing platform for smaller but also for very large computing platforms (e.g. exascale computing).
- ***Modeling parallel and serial modules in the application***
  Depending on the application semantics, while some modules can run in parallel others can only run in serial. Typically serial modules will be mapped to a single node, while parallel modules need to be mapped to multiple nodes.
- ***Modeling the mapping of parallel modules to physical nodes***
  The DSL should provide mechanisms for describing the allocation. The mapping of the modules to the computing platforms can be done in different ways.
- ***Defining the interaction patterns among parallel modules***
  Parallel modules will typically exchange information to perform the requested tasks. In general it is important to define the proper interaction patterns not only for functional reasons but also to optimize the parallelization overhead and as such increase efficiency.

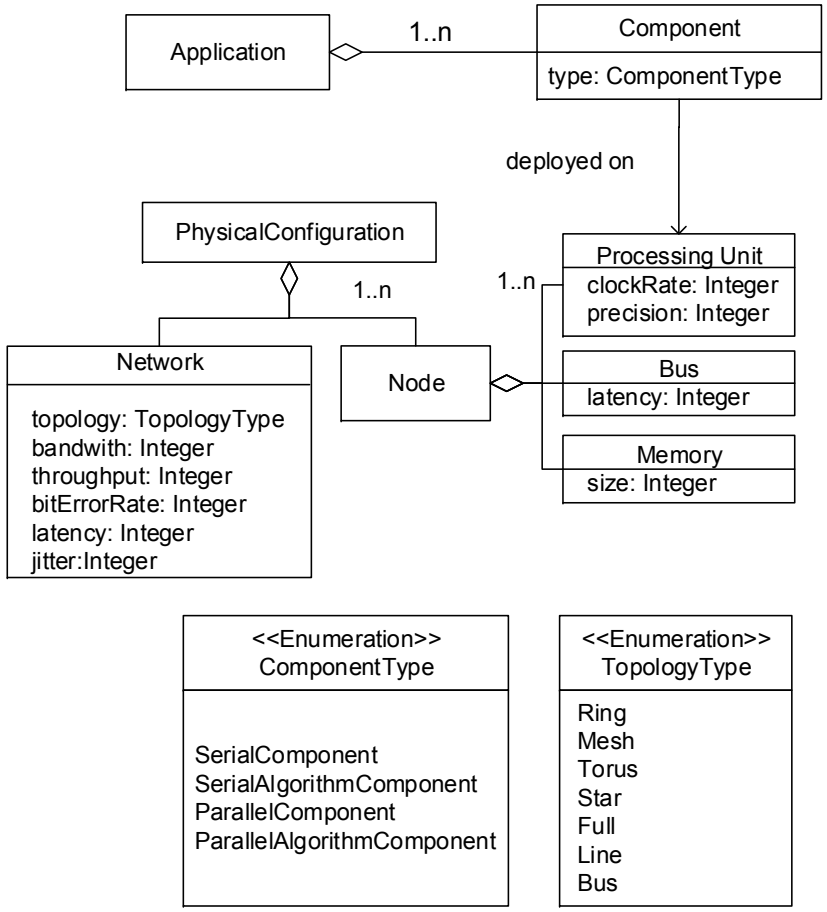
```
Procedure Complete-Exchange:
for i=1 to n-1
   Collect data to the dominating nodes from the dominated nodes
Endfor
for i=n-1 downto 1
   Exchange the selected data between dominating nodes
   Distribute data from dominating nodes to the dominated nodes
Endfor
```



Parallel Computing Platform (Physical Configuration)

Deployment Metamodel

- The viewpoint is based on a metamodel that is derived after a domain analysis on parallel computing

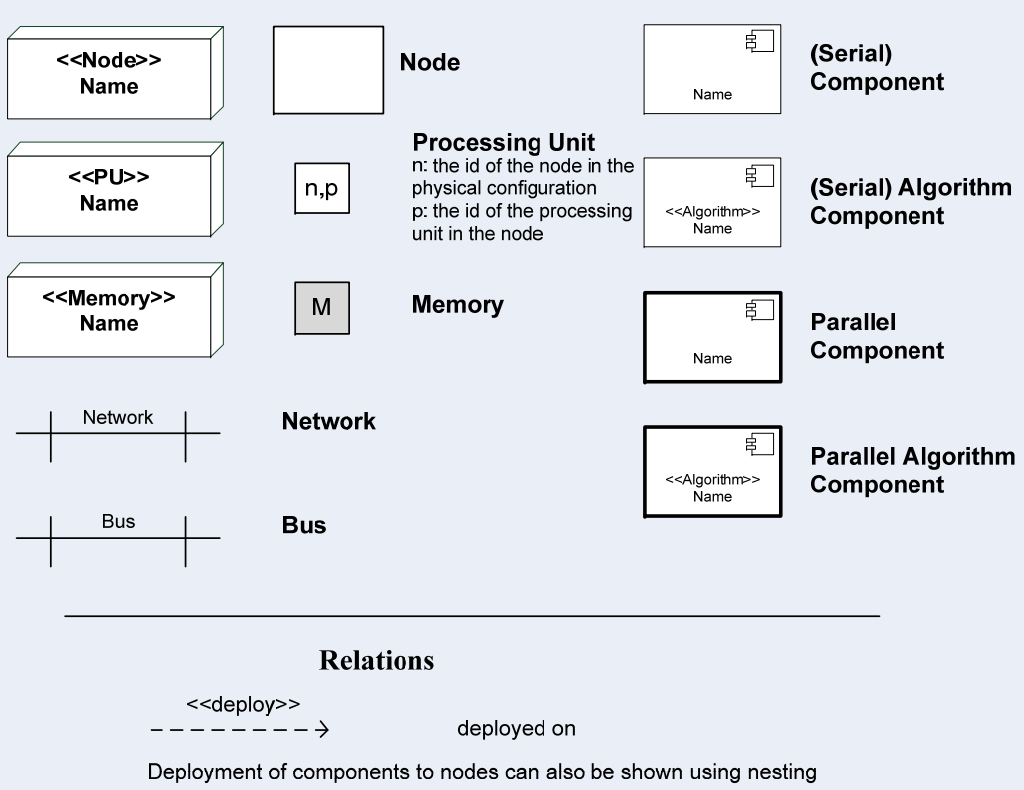


Deployment Viewpoint

Section	Description
«Viewpoint Name»	Deployment Viewpoint
«Overview»	The deployment for the components of the parallel application
«Concerns»	Which component runs on which processing unit?
«Typical Stakeholders»	System Engineer
«Constraints »	Parallel component can be deployed on different processing units. Serial component can be deployed on a single processing unit.

Deployment Viewpoint – Cnt'd

«Model types and notation»



Model-Driven Software Development

Parallel Computing- Observation

Lack of Precise Models

- In current parallel computing approaches **standard modeling approaches for supporting the mapping process seems to be lacking**.
- Most approaches seem to adopt more **informal conceptual 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.

Manual Process

- The mapping process is **largely a manual process** and needs better automation support

Model-Driven Software Development

- Model-*Driven* Software Development adopts models as the basic abstraction.
- Models **do not constitute documentation**,
- but are **considered equal to code**
- as their implementation is **automated**.

Maturity of Models...

- **Sketch** – simple drawing model; not precise or complete, nor is it intended to be. The purpose of the sketch is to try out an idea. The sketch is neither maintained nor delivered.
- **Blueprint** – document/design model describing properties needed to build the real thing. In other words, the blueprint is the embodiment of a plan for construction
- **Executable** – software model that can be compiled and executed; can be automatically translated into other models or code

Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principle of Model Driven Architecture. Addison Wesley, Reading , 2004.

Model – Software Engineering - Definition

- "**A model is a description of a (part of) system written in a well-defined language**.

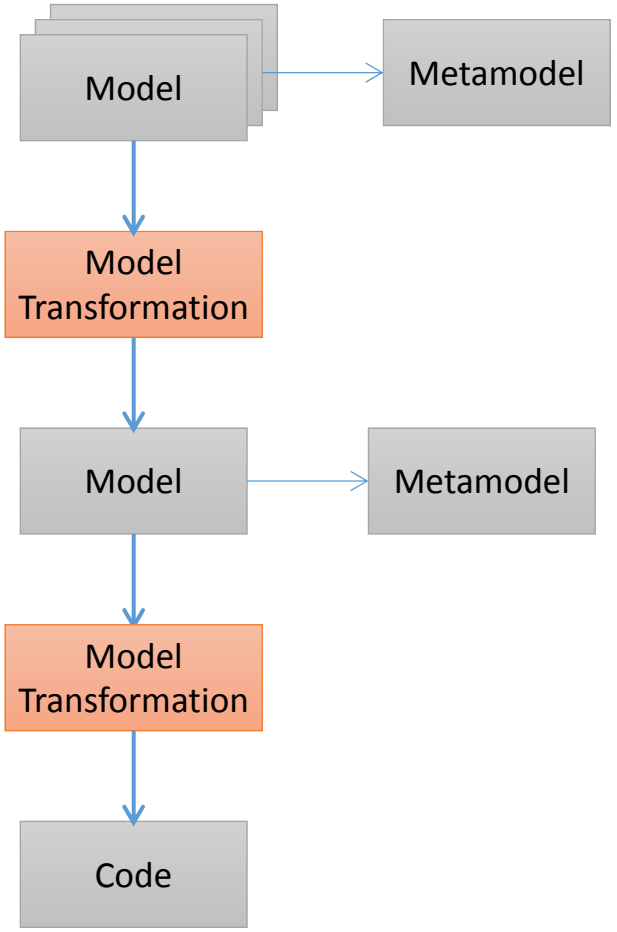
- A well-defined language is a language with well-defined form (syntax), and meaning (semantics),

- which is suitable for automated interpretation by a computer"

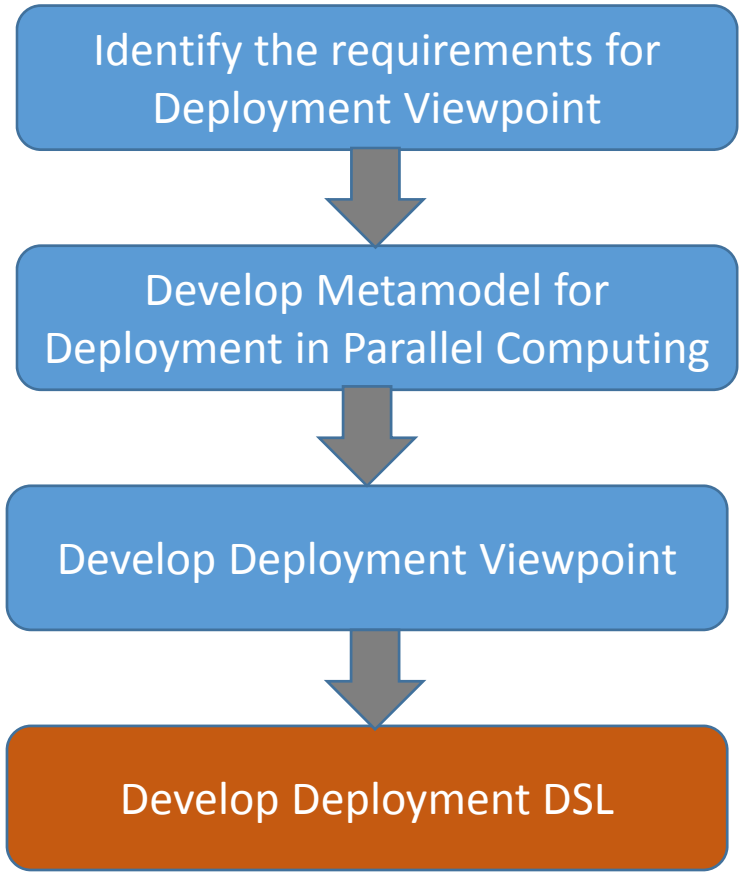
A. Kleppe, S. Warmer, W. Bast, "MDA Explained. The Model Driven Architecture: Practice and Promise", Addison-Wesley, April 2003

How does MDSD work?

- Developer develops **model(s)** based on certain metamodel(s).
- Using **code generation (model-to-text transformation)**, the model is transformed to executable code.
- Optionally, the **generated code is merged** with manually written code.
- One or more **model-to-model transformation steps** may precede code generation.



Approach for Developing DeploymentViewpoint/ DSL for Mapping Process

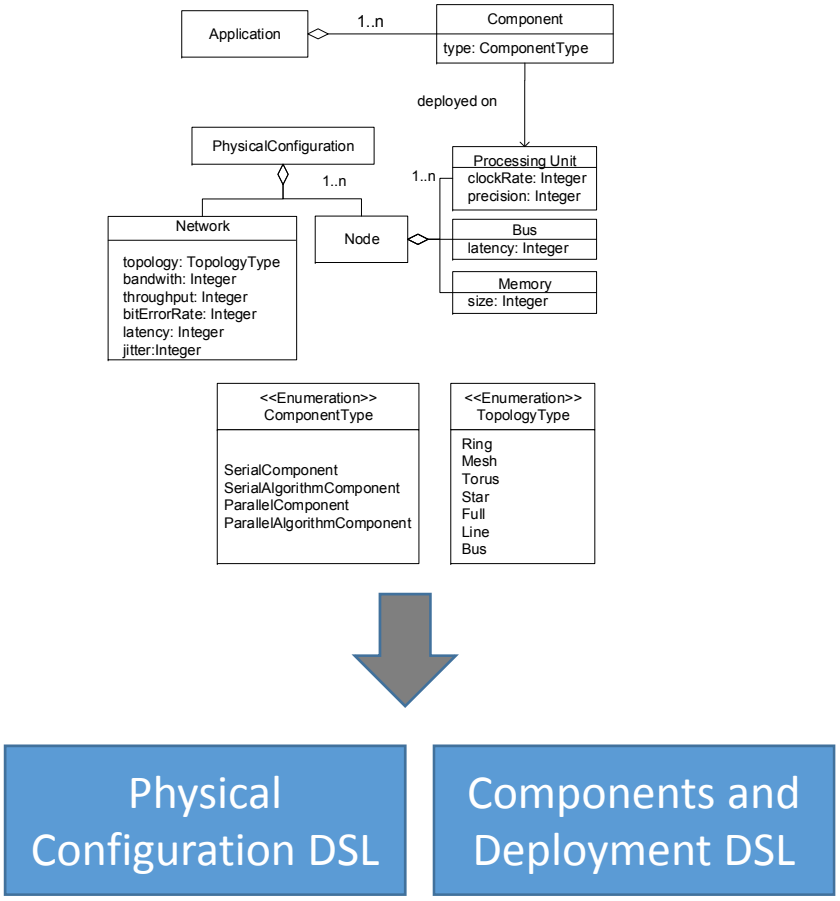


Blueprint

Executable

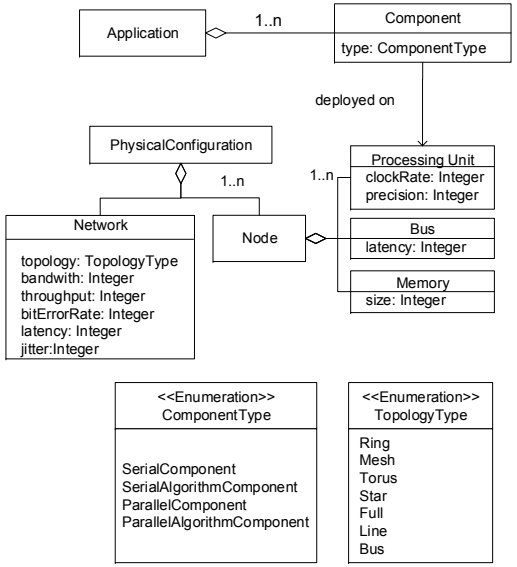
Deployment DSL

- In principle we could describe one DSL that implements all the required concerns for deployment.
- However, we have decided to define two separate DSLs including a DSL for describing the physical configuration and a DSL for describing the components and the deployment of these components on the nodes.
- In this way both concerns can be described separately. Further, the same physical configuration can be used with different deployment descriptions.



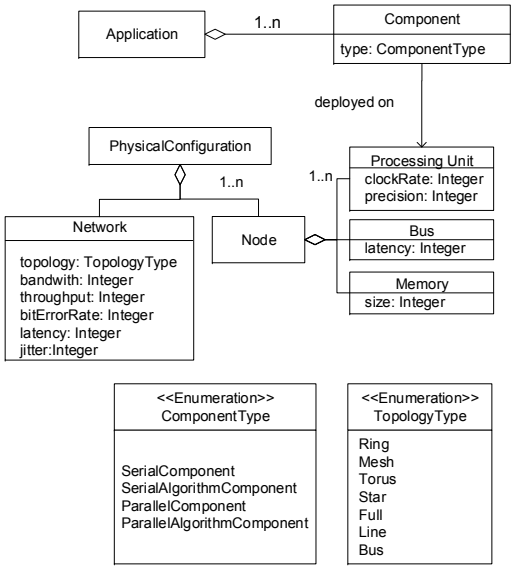
Physical Configuration DSL

```
1.  grammar PhysicalConfiguration
2.  Model:
3.   types+=TypeDef*
4.   physicalConfigurations+=PhysicalConfiguration*;
5.  PhysicalConfiguration:
6.   'physicalconfiguration' name=ID '{'
7.     'nodes' ':' nodes+=[NodeType] '['size=INT']'
     (',' nodes+=[NodeType] '['size=INT']')* ';'
8.     'network' ':' network=[NetworkType] ';'
9.   '}';
10. TypeDef:
11.  NodeType | MemoryType | BusType |
12.  ProcessingUnitType | NetworkType;
13. NetworkType:
14.  'network' name=ID '{'
15.    'topology'':' topology=TopologyType ';'
16.    'bandwith'':' bandwith=INT bunit=BwithUnit';'
17.    'throughput'':' throughput=INT ';'
18.    'bitErrorRate'':' berate=INT '%' ';'
19.    'latency'':' latency=INT 'usec' ';'
20.    'jitter'':' jitter=INT ';'
21.  '}';
22. enum TopologyType:
23.  Ring='Ring'|Mesh='Mesh'|Torus='Torus'|
     Star='Star'|
24.  Full='Full'|Line='Line'| Bus='Bus';
25. NodeType:
26.  'node' name=ID '{'
27.    'processingunits'':' pus+=[Pro-
     cessingUnitType] '['size=INT']' (',' pus+=[Pro-
     cessingUnitType] '['size=INT']')* ';'
28.    'memory' ':' memory=[MemoryType] ';'
29.    'bus' ':' bus=[BusType] ';'
30.  '}';
31. ProcessingUnitType:
32.  'processingunit' name=ID '{'
33.    'clockRate'':' rate=DOUBLE unit=ClockRateType
     ';'
34.    'precision'':' precision=PrecisionType ';'
35.  '}';
36. MemoryType:
37.  'memory' name=ID '{'
38.    'size' ':' size+=MemorySize ';'
39.  '}';
40. MemorySize: size=INT unit=MemorySizeUnit;
41. DOUBLE returns EDouble: '-'? INT? '.' INT;
```



Deployment DSL

```
1.  grammar Deployment
2.  import "/dsl/PhysicalConfiguration" as pc
3.  Model: applications+=Application*;
4.  Application:
5.   'application' name=ID '{'
6.     components+=Component*
7.   '}';
8.  Component:
9.   'component' name=ID '{'
10.    'type'':' type=ComponentType ';'
11.    'deployed on'':' pus+=DeploymentUnit
12.                     (',' pus+=DeploymentUnit)*';'
13.  '}';
14. DeploymentUnit:
15.  physicalconfiguration+=
     [pc::PhysicalConfiguration]('.' 'nodes'
     '['(index=INT | fromIndex=INT '..'
     toIndex=INT)(','(index=INT | fromIndex=INT '..'
     toIndex=INT))*']');
16. enum ComponentType:
17.  serial='SerialComponent'|
18.  parallel='ParallelComponent' |
19.  serialalg='SerialAlgorithmComponent' |
20.  parallelalg='ParallelAlgorithmComponent';
```



Case Study – Traffic Simulation

- To illustrate the DSL we use a case study for the development of a traffic simulation
- The goal of this simulation is to support the analysis and optimization of various traffic flow parameters for efficient movement of traffic and minimal traffic congestion problems.
- Typically, a traffic simulation consists of a large set of components that need to be deployed on a parallel computing platform.
- The main components of the simulation environment are cars, trucks, drivers, speed cameras, traffic lights, lane closes and a traffic analyzer.

Simulation Component	Number
Car Simulation	600
Truck Simulation	80
Driver Simulation	680
Speed Camera Simulation	5
Traffic Light Simulation	15
Lane Close Simulation	4
Traffic Analyzer Simulation	1

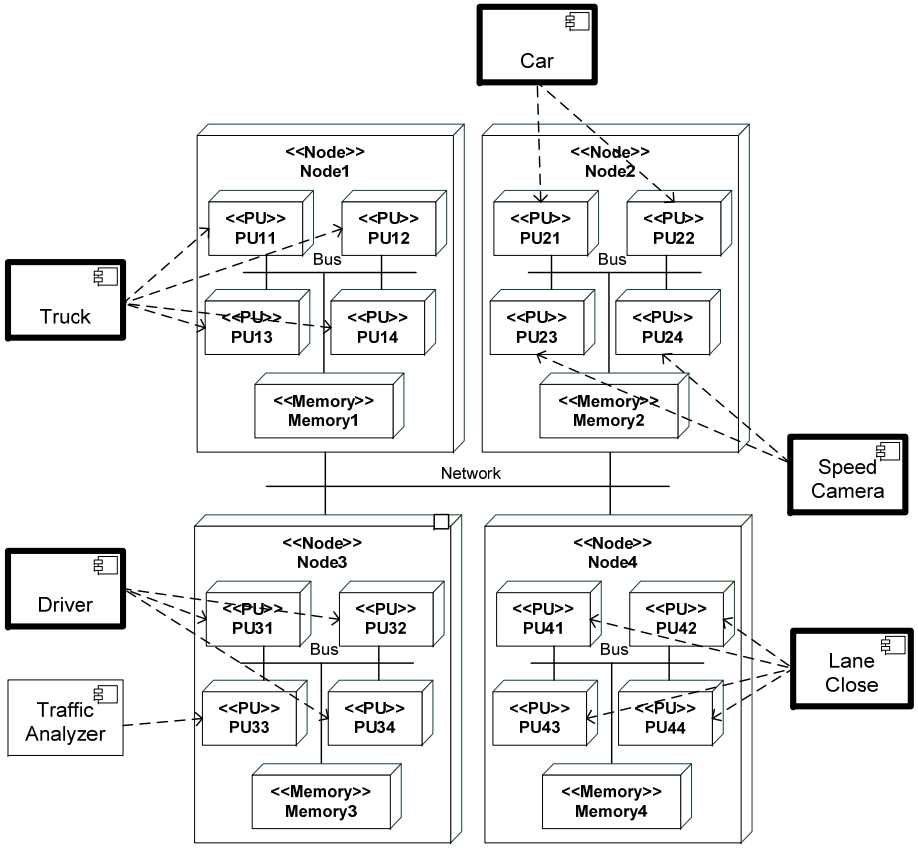
Deployment View Example - Traffic Simulation

The Physical Configuration of the Simulator Computer is constructed using 4 nodes.

Each node includes 4 processing units and a memory.

Nodes are connected using a network.

On this physical configuration, the components of the Traffic Simulator application are deployed on processing units using deployment relations



Traffic Simulator Computer Physical Configuration

- For the provided scenario assuming that all the components need to be deployed on a distinct processing unit, then at least 1385 processing units will be needed.
- The specification on the right shows an example Traffic Simulator Computer physical configuration consisting of 350 Traffic Simulator Host nodes and a Traffic Simulator Network.
- Each node includes 4 PowerPC 450 processing units, so the physical configuration includes **1400 PowerPC 450 processing units.**
- The PowerPC 450 processing unit is a 850 MHz core with 32 Bit precision.
- Each node has a 4 GB DDR2 memory and a bus with 300 usec latency.
- The network is constructed as Torus topology with 1 GBit band-with, 20% bit error rate and etc.

Simulation Component	Number
Car Simulation	600
Truck Simulation	80
Driver Simulation	680
Speed Camera Simulation	5
Traffic Light Simulation	15
Lane Close Simulation	4
Traffic Analyzer Simulation	1

```
1.  memory DDR2Memory {
2.   size : 4 GByte;
3.  }
4.  processingunit PowerPC450 {
5.   clockRate : 850.00 MHz;
6.   precision : 32Bit;
7.  }
8.  bus PowerPcBus {
9.   latency : 300 usec;
10. }
11. node TrafficSimulatorHost {
12.  processingunits : PowerPC450[4];
13.  memory : DDR2Memory;
14.  bus : PowerPcBus;
15. }
16. network TrafficSimulatorNet {
17.  topology : Torus;
18.  bandwith : 1 GBit;
19.  throughput : 10;
20.  bitErrorRate : 20%;
21.  latency : 500 usec;
22.  jitter : 10;
23. }
24. physicalconfiguration TrafficSimulatorComputer {
25.  nodes : TrafficSimulatorHost[350];
26.  network : TrafficSimulatorNet;
27. }
```

Traffic Simulator Computer Deployment

- To define the deployment of the Traffic Simulator application on Traffic Simulator Computer, the components of the application are defined including 'deployed on' relation.

- For example Car components are deployed on node 1 to 150, which means they are deployed on 600 (4 x 150) processing units.

```
1.  application TrafficSimulator {
2.   component Car {
3.     type : ParallelComponent;
4.     deployed on :
     TrafficSimulatorComputer.nodes[1..150];
5.   }
6.   component Truck {
7.     type : ParallelComponent;
8.     deployed on :
     TrafficSimulatorComputer.nodes[151..170];
9.   }
10.  component Driver {
11.    type : ParallelComponent;
12.    deployed on :
     TrafficSimulatorComputer.nodes[171..340];
13.  }
14.  component SpeedCameraSim {
15.    type : ParallelComponent;
16.    deployed on :
     TrafficSimulatorComputer.nodes[341],
     TrafficSimulatorComputer.nodes[342].processingu
     nits[1];
17.  }
18.  component TraficLightSim {
19.    type : ParallelComponent;
20.    deployed on :
     TrafficSimulatorComputer.nodes[342].processingu
     nits[2..4],
     TrafficSimulatorComputer.nodes[343..345];
21.  }
22.  component LaneCloseSim {
23.    type : ParallelComponent;
24.    deployed on :
     TrafficSimulatorComputer.nodes[346];
25.  }
26.  component TrafficAnalyzer {
27.    type : SerialComponent;
28.    deployed on :
     TrafficSimulatorComputer.nodes[347].processingu
     nits[1];
29.  }
30. }
```

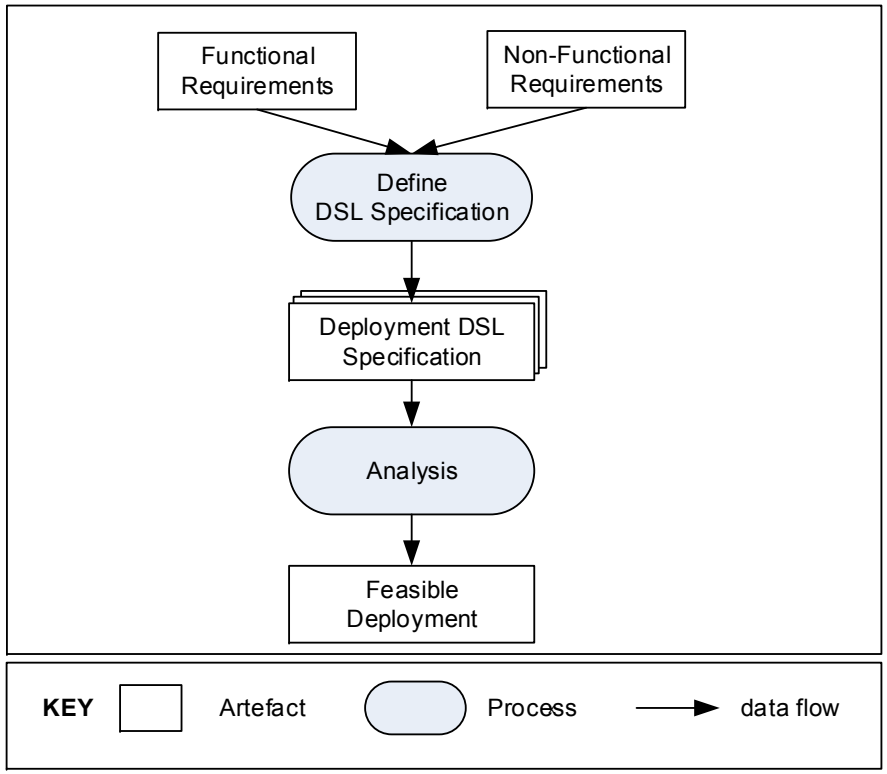
Possible Future Research Directions

Important Scenarios

- Analysis of Deployment Alternatives
- Automated Deployment of Application Code to Computing Platform
- Documentation of Deployment Architecture

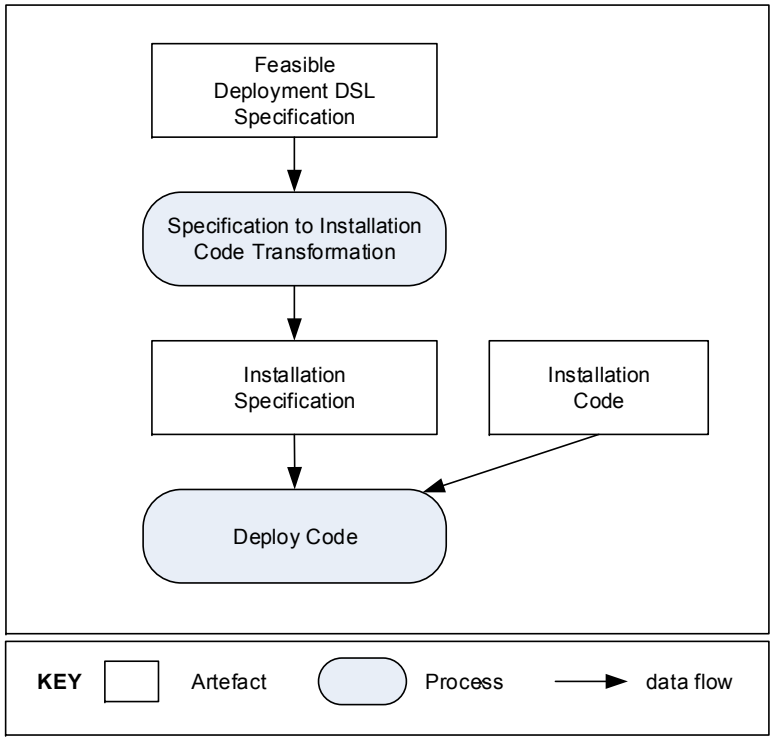
Analysis of Deployment Alternatives

- Using the DSL the deployment alternatives can be specified for a given physical configuration.
- The physical configuration components have attributes that affect the performance evaluation of the parallel application.
  - For instance, the latency of the network increases the messaging overhead of the parallel application which will increase the total elapsed time of the application processes.
- Thus, each specification could be analyzed with respect to the defined functional and non-functional requirements. Using the DSL, different specifications can be defined that describe different alternatives.

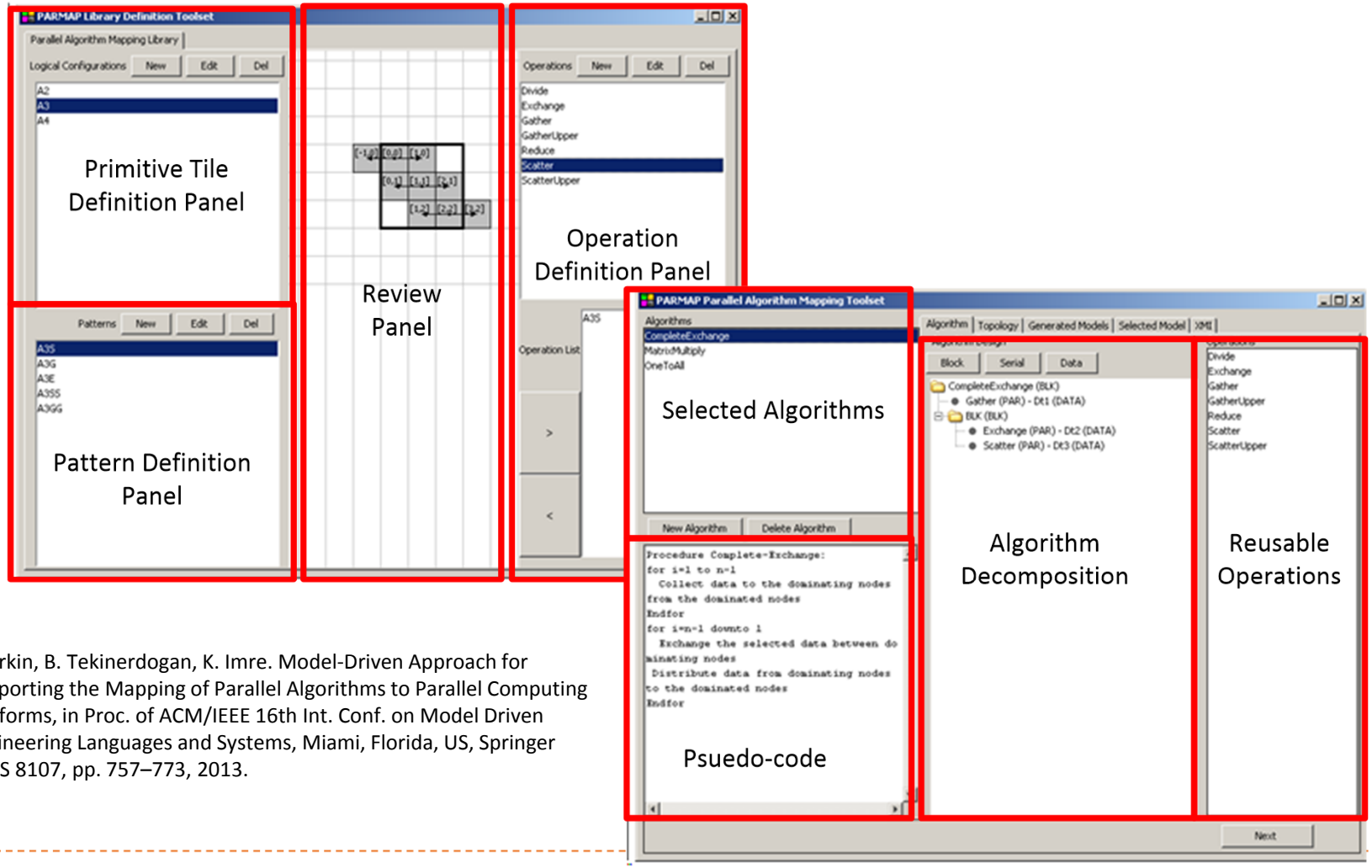


Automated Deployment of Application Code to Computing Platform

- The application code needs to be deployed on the computing platform to be run in parallel.
- In the literature, various tools can be found which concern the automatic deployment of the code to the nodes of a parallel computing platform.
- These tools mainly use installation specifications or configurations that define how the deployment framework will distribute and run the tasks among the processing units.
- To support reuse and integrate the design and analysis activities with the eventual installation of the code, the DSL specification of the feasible deployment alternative can be transformed to the format required by the existing tools.
- Based on the feasible deployment DSL specification the installation specification will be generated.
- Given the installation case, which we assume is automatically generated as well, the application can then be deployed automatically. In particular for large scale parallel computing platform this overall process will pay off and increase productivity and quality.



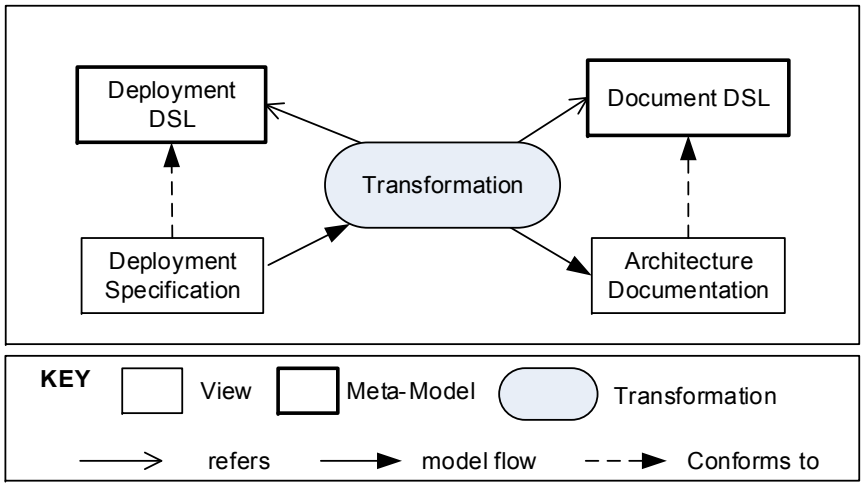
ParMapper



E. Arkin, B. Tekinerdogan, K. Imre. Model-Driven Approach for Supporting the Mapping of Parallel Algorithms to Parallel Computing Platforms, in Proc. of ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems, Miami, Florida, US, Springer LNCS 8107, pp. 757–773, 2013.

Documentation of Deployment Architecture

- Every architecture needs a documentation to guide architecture stakeholders about how to benefit from the architecture and clarify ambiguous points.
- Architecture documentation is a communication artifact for all stakeholders and it is used during the whole lifecycle of the architecture.
- It contains both natural language descriptions about system and formal architecture models.
- We utilize our DSLs in order to automatically generate the architecture view related part of the architecture documentation. The generation is done via model-to-text transformation.



Recent Related Papers

- H.G. Gürbüz, N. Pala Er, B. Tekinerdogan. Architecture Framework for Software Safety, to be published in Proc. of the 8th System Analysis and Modelling Conference (SAM'14), Valencia, Spain, September 29-30, 2014.
- E. Arkin, B. Tekinerdogan, K. Imre. Model-Driven Approach for Supporting the Mapping of Parallel Algorithms to Parallel Computing Platforms, in Proc. of ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems, Miami, Florida, US, Springer LNCS 8107, pp. 757–773, 2013.
- T. Celik, O. Koksal, B. Tekinerdogan. Deploy-DDS: Tool Framework for Supporting Deployment Architecture of Data Distribution Service based Systems , to be published in Proc. of the 8th European Conference on Software Architecture (ECSA 2014), Vienna, Austria, 2014.
- E. Arkin, B. Tekinerdogan. Model-Driven Transformations for Mapping Parallel Algorithms on Parallel Computing Platforms, in Proc. of Model-Driven Engineering for High Performance and Cloud Computing workshop, Models 2013, Miami September 29, 2013.
- B. Tekinerdogan, E. Arkin. Architecture Framework for Mapping Parallel Algorithms to Parallel Computing Platforms, in Proc. of Model-Driven Engineering for High Performance and Cloud Computing workshop, Models 2013, Miami September 29, 2013.
- T. Çelik, B. Tekinerdogan, K. Imre. Deriving Feasible Deployment Alternatives for Parallel and Distributed Simulation Systems, ACM Transactions on Modeling and Computer Simulation journal, Volume 23 Issue 3, pp. July 2013
- T. Çelik, B. Tekinerdogan. S-IDE: A Tool Framework for Optimizing Deployment Architecture of High Level Architecture Based Simulation Systems. Elsevier Journal of Systems and Software, Volume 86, Issue 10, October 2013, pp. 2520–2541, 2013.
- ….

Conclusion

- We have provided a systematic model-driven approach for mapping parallel algorithms to parallel computing platforms
- First we have developed the required metamodel
- Based on the metamodel we have proposed a deployment viewpoint for mapping parallel applications to parallel computing platforms
- A domain specific language (DSL) has been implemented based on the viewpoint
- The DSL as such is expressive and provides a declarative and scalable approach for representing different deployment views in parallel computing.
- Using the DSL, the deployment of applications to even very large computing platforms such as exascale computing can be conveniently modeled.
- In addition, the executable DSL specifications can be used to support
  - the automated analysis,
  - automated generation of deployment specification, and
  - automated generation of deployment architecture documentation.
- In our future work we aim to elaborate on these different important scenarios for supporting generative parallel computing.

Acknowledgement

- Many ideas in this presentation are the result of joint research

Thanks to:

- Ethem Arkin, Aselsan/Hacettepe University
- Kayhan Imre, Hacettepe University