Computational Orchestrator: A Super Class for Matrix, Robotics and Control System Orchestration

Shweta Agrawal
Lecturer, SCSIT, DAVV,
Indore, 452001 India

Raj Kamal
Professor, MIST,
Indore, 453331 India

ABSTRACT
Orchestrator is a middleware which coordinates the sequence of execution of distributed tasks. Numerous Orchestrators are available for different applications. A composite Orchestrator for the orchestration of matrix operations, robotic tasks and control systems is does not exist. The paper proposes the design and implementation of a composite Orchestrator. The proposed Orchestrator is orchestrating matrix operations, robotic tasks and tasks in control systems. It is implemented as a super class of the different Orchestrators. Multithreading and RMI in Java is used to implement the Orchestrator.

Keywords
Orchestrator, matrix operations, robots, NCS.

1. INTRODUCTION
Distributed computing has a number of advantages. It reduces time complexity. It enables modular programming. It enables reusability of components. The distributed computing is applicable in almost all fields of engineering and science. It includes web based applications, robotics and control systems. An Orchestrator [1] [2] handles numerous issues of distributed computing. The issues in distributed computing are synchronization, timeouts, priority, failure of nodes.

There are numerous Orchestrators for orchestrating the WSs and robotic tasks. Table 1 gives the description of earlier proposed models for Orchestrators. The Orchestrators described in the table are specifically designed for the WSs, cloud computing or robotic tasks. The table also gives the limitations of the earlier proposed models. The earlier proposed models do not describe the timing relations in the orchestration steps.

Table 1. Models of Orchestrator

S.No.	Models Presented	Features	Limitation
1	Orchestrator model for WSs [3] (Figure 1)	• Orchestration of the n number of functioning WSs. • Orchestrator invokes and WSs respond. • Orchestrator decides the invoking sequence of WSs according to the application.	• Applicable to WSs. • Timing relations are not known.
2	Service Orchestrator for improving the QoS [4] (Figure 2)	• Several services may provide the same functionality, with different levels of performance and reliability, and at different costs. • Figure 3.2 shows $m \geq 1$ sets of concrete services: The set CSi $=\{s_i^1, s_i^2, - - - s_i^{n_i}\}$, $1 \leq i \leq m$, is comprised of $n_i \geq 1$ concrete services that provide the same abstract service.	• Applicable to WSs. • Timing relations are not known.
3	Orchestrator for WSs handling [5] (Figure3)	• The orchestration is performed using XML schema. • Figure 3.3 shows the model. Four types of orchestration execution are: ➢ Sequential ➢ Parallel ➢ Conditional ➢ Looping	• Applicable to WSs. • Timing relations are not known.

4	Polyphony: Orchestration Framework for Cloud Computing [6]	• A component based framework, works for cloud computing. • OSGi framework is used to implement the Polyphony.	• Applicable to cloud computing.
5	Orchestrator for robotic tasks [7]	• Framework uses object oriented approach. • General Command Interpreter decomposes a command into sub commands. • Subcommands are further decomposed into sub subcommands. • Process continues until the output becomes a sequence of robot primitive actions.	• Applicable to robotic tasks. • Timing relations are not known.

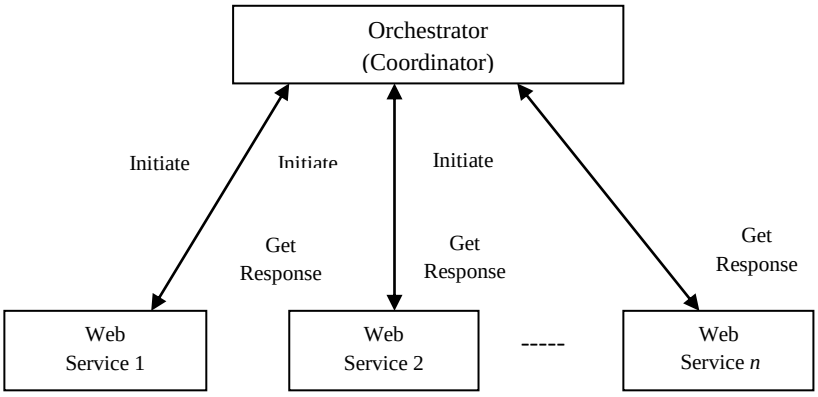

Fig 1: Functioning of Orchestrator for the web services [3]

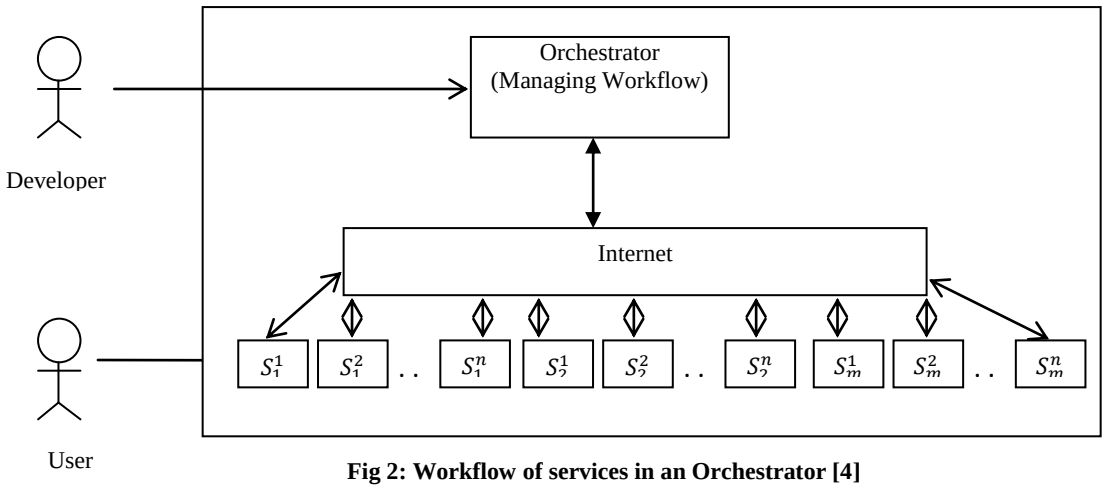

Fig 2: Workflow of services in an Orchestrator [4]

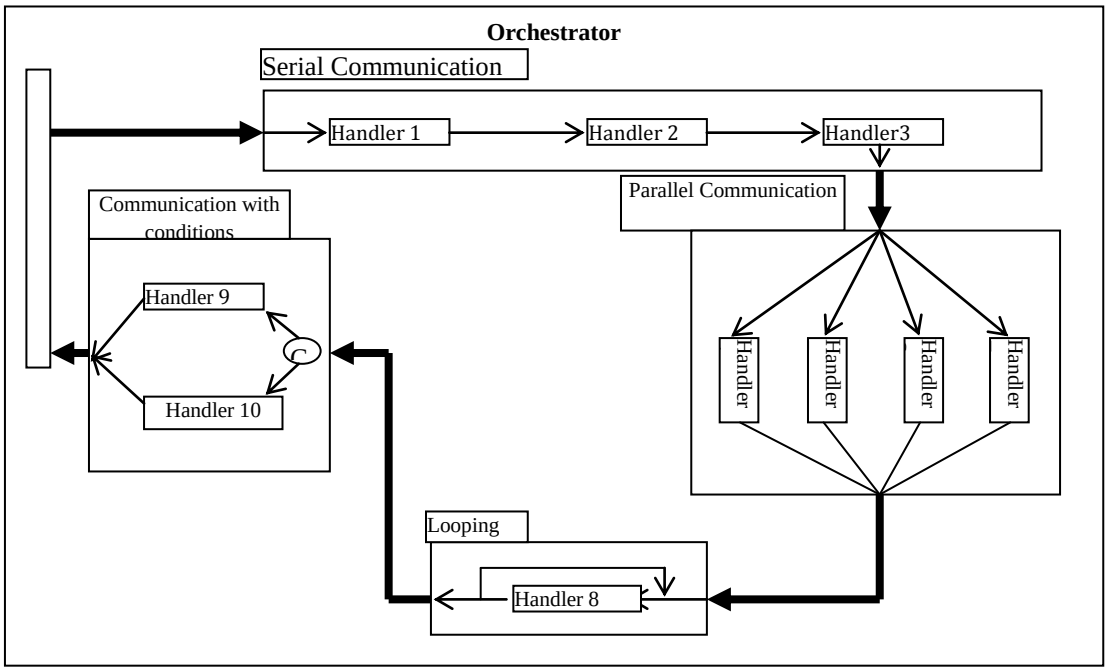

Fig 3: An Orchestrator framework for WS handlers [5]

Matrix operations are fundamental of many mathematical calculations and require in problems of science and engineering. Matrix multiplication and matrix inversion are the most frequently used matrix operations. An Orchestrator [1] [2] can be designed to find faster solutions of matrix operations. There is no any Orchestrator model to orchestrate the matrix operations.

To speedup matrix multiplication, there has been a great deal of interest in developing parallel formulations of various algorithms. There are many parallel architectures [8-10] available for speeding up the computation.

Networked Control Systems (NCS) consists of sensors, actuators and controllers [11] .NCS are used in automation of distributed systems. Orchestrator can be used to automate the functioning of NCS. No any models of Orchestrator are proposed in this area.

A common Orchestrator architecture is required, which can orchestrate the matrix operations, robotic tasks and control systems. The paper proposes a common Orchestrator architecture named as Computational Orchestrator (CompOrch). CompOrch orchestrates the matrix operations, robotic task and application of control systems.

2. CompOrch ARCHITECTURE

Figure 4 shows the layered architecture for CompOrch. Application user is on the upper most layers. Application user can choose any Orchestrator to work with. The job of CompOrch is to coordinate with various Orchestrators. Various Orchestrators include IOrch, IOrchE, MOrch, ROrch and COrch. The introduction of these Orchestrators is given in next section.

CompOrch creates the object of specified Orchestrator and invokes the controlling method of the Orchestrator. Once the object has been created and controlling method has been called, the individual Orchestrator grabs the control of execution.

The called Orchestrator establishes the connection with remote services. The job of called Orchestrator is to coordinate with various events. The events may be termed as *to send signal, to receive signal, to generate control code* according to user specification. The last layer is of actual service or hardware. The CompOrch integrates and coordinates the actual service or hardware.

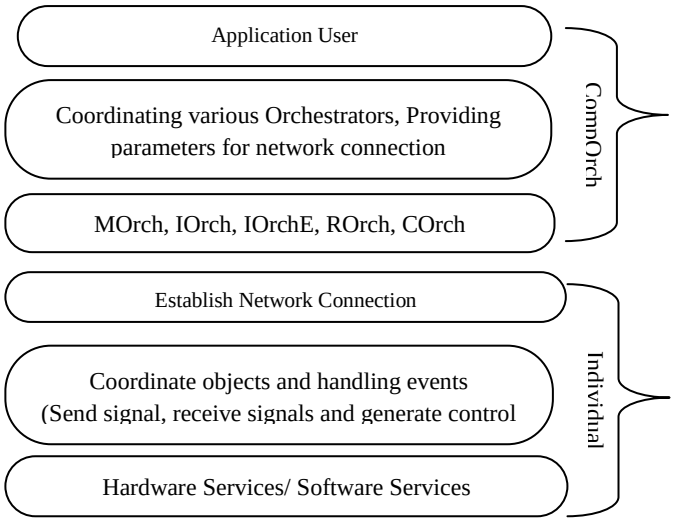

Fig 4: Layered architecture of CompOrch

3. INTRODUCTION TO ORCHESTRATORS EXTENDED FROM THE CompOrch

CompOrch extends to five Orchestrators. The extended Orchestrators are as follows:

IOrch finds the inverse of a given square matrix through Cramer's rule. IOrch orchestrates two services. One for calculating determinant of matrix, other for calculating minor

matrix for the given row and column number. IOrch orchestrates the sequence of calculating minor and determinants.

IOrchE is an extension of IOrch. Four services are implemented in IOrchE. As calculating determinant is more time consuming as compare to calculating minor of matrices. One service is implemented for minor and three services are implemented for determinant. IOrchE Orchestrates the execution sequence of these four services for calculating the inverse of a given matrix.

MOrch orchestrates the matrix multiplication. The algorithm used for matrix multiplication is Strassen's algorithm. Seven servers are implemented. The servers execute the seven equations of Strassen's algorithm. The task of MOrch is to orchestrate all the seven services to find the multiplication of two square matrices.

ROrch orchestrates multiple moving robots. Robots can move in any direction and in any order. Robots can be invoked in sequential, parallel or in hybrid manner. User can enter the codes for deciding the sequence and direction of movement.

COrch shows the design of Orchestrator to handle the software complexity of NCS. The COrch orchestrates an example of NCS. The example taken is Collision detection and avoidance system in multiple moving robots. COrch also decides the invoking sequence of robots from sequential, parallel or hybrid Execution.

4. GENERAL ARCHITECTURE OF CompOrch

Figure 5 shows a general architecture of CompOrch. It is based on a service based system described by C. Grunske *et al.* [4]. Assume *m* sets of services for orchestration. Assume that each set has *p, q* or *r* number of services. The Orchestrator has *k* stages of services. An extended Orchestrator overrides the values of *p, q, r, m* and *k as* per the workflow pattern. The earlier described system [4] did not represent the timings of execution of each service. CompOrch shows the timings of execution of each service.

Assume t_1, t_2...t_n are the initiation time of different tasks. t_1 is the start orchestration time .t_2, t_3... t_n are the initiation timings at stages 1, 2,…. *k*.

The time relation $t_2 < t_3 <$... t_n says that at one stage all services are initiated in parallel and at different stages services are initiated in sequential manner.

Similarly, $t_2 = t_3 =$ ... t_n, means all stages are initiated in parallel manner.

Table 2 gives the meaning of symbols used in architecture of CompOrch.

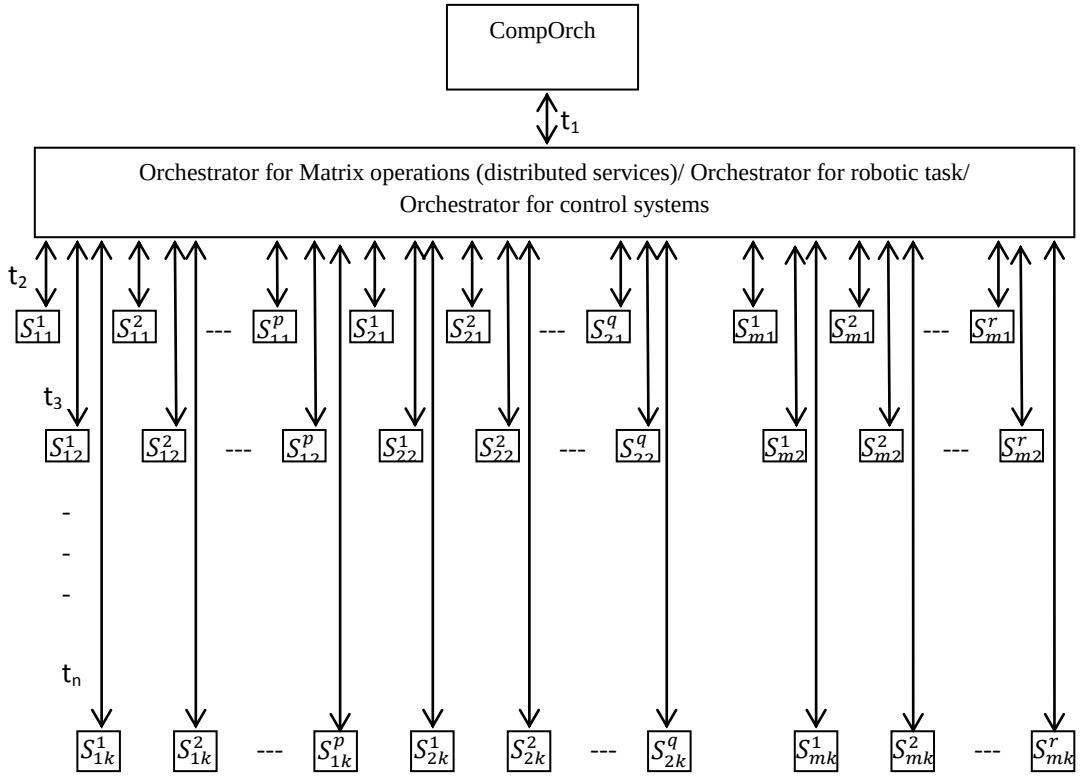

Fig 5: General architecture of CompOrch

Table 2. Symbol used in general architecture of CompOrch

Symbol used	Meaning
S_{11}^{1}	First service of set one at stage one
S_{11}^{p}	p[th] service of set one at stage one
S_{21}^{q}	q[th] service of set two at stage one
S_{m1}^{r}	r[th] service of set *m* at stage one.
S_{12}^{1}	First service of set one at stage two
S_{12}^{p}	p[th] service of set one at stage two
S_{22}^{q}	q[th] service of set two at stage two
S_{m2}^{r}	r[th] service of set *m* at stage two.
S_{1k}^{1}	First service of set one at stage *k*
S_{1k}^{p}	p[th] service of set one at stage *k*
S_{2k}^{q}	q[th] service of set two at stage *k*
S_{mk}^{r}	r[th] service of set m at stage *k*

5. ComputationalOrchestrator: A SUPER CLASS

ComputationalOrchestrator is a super class of described five Orchestrators. All five Orchestrators are extending the ComputationalOrchestrator class. The Orchestrators are using a specific interface to connect with the remote machine.

As all Orchestrators are using RMI for communicating with remote machines. So the CompOrch provides the parameters for functioning with the RMI. The Orchestrators are overriding the parameters of CompOrch.

Figure 6 shows the extension of CompOrch for the distributed computing. Matrix operations are implemented to represent the distributed computing.

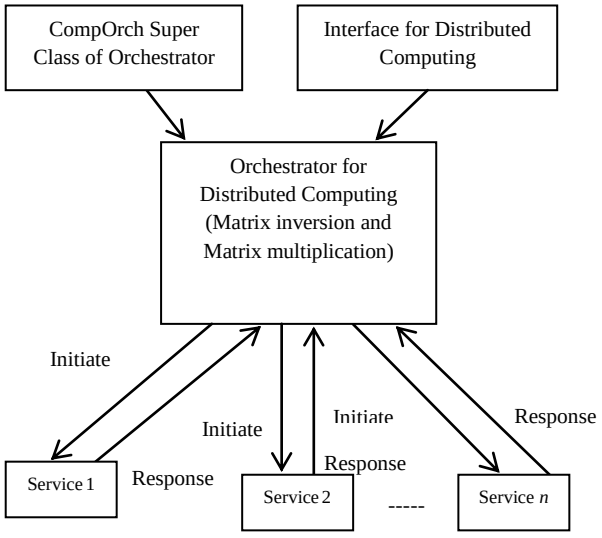

Fig 6: CompOrch's extension for distributed computing

Figure 7 shows the extension of CompOrch for Robots. The Orchestrator for robotic tasks extends the

CompOrch. There are *n* robots. Orchestrator sends and receives signals. It then controls the movement of robots.

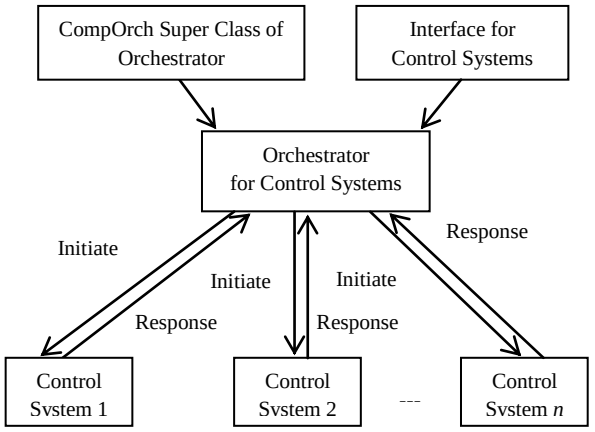

Fig 7: CompOrch's extension for controlling movement of robots

Figure 8 shows the extension of CompOrch for control systems. The *n* control systems may be there, Orchestrator controls the functioning of multiple control systems by initiating messages in sequential, parallel or in hybrid manner. CompOrch provides the network communication parameters to the control system Orchestrator.

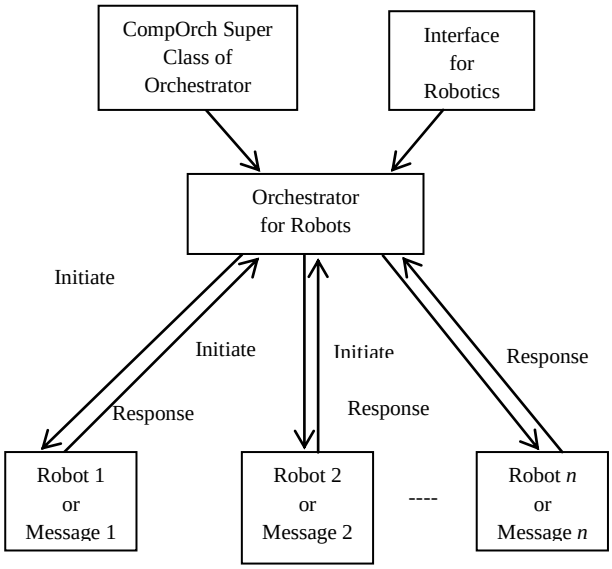

Fig 8: CompOrch's extension for control systems

6. IMPLEMENTATION

The CompOrch is implemented using Java multithreading and RMI. Following steps describe the workflow in CompOrch:

Step1: All servers are RMI server. They are binding their lookup name with an RMI registry. The servers execute various services. The services are determinant, minor and robotic movement.

Step2: User gives the input to the CompOrch for the type of orchestration, whether Mathematical orchestration, robotic orchestration or control system orchestration.

Step3: CompOrch creates the object of specified Orchestrator.

Step4: CompOrch invokes the method of specified Orchestrator.

Step5: The specified Orchestrator is an RMI client and extending to CompOrch. It makes connection with the appropriate server by matching the lookup name.

Step6: The specified Orchestrator asks to the user about mode of execution and the appropriate data for orchestration.

Step7: The specified Orchestrator starts the mentioned execution (sequential, parallel or hybrid) after taking inputs from user.

Step8: The specified Orchestrator displays the results after completing the mentioned orchestration.

Step9: CompOrch is ready to accept another type of orchestration.

6.1 Class Diagram For CompOrch

Figure 9 shows the class diagram for the CompOrch.

- The main class is the ComputationalOrchestrator class .It provides interface to user. The class ComputationalOrchestrator represents to the CompOrch.
- The ComputationalOrchestrator class uses four main classes of four Orchestrators.
- RMIClientMath for IOrch and IOrchE.
- StrassenClient for MOrch.
- RoboClient for ROrch.
- ControlClient for COrch.

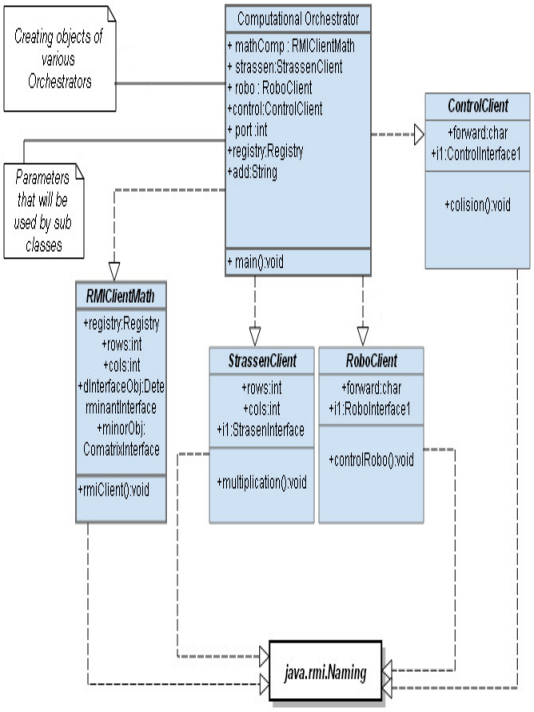

Fig 9: Class Diagram for CompOrch

The described classes of Orchestrators are using RMI servers for initiating distributed node operations. All four classes are using the java.rmi.Naming class to search the server with the help of lookup name of server. There are several attributes and methods in represented classes, only few of them are shown in the diagram to keep the diagram simple. Table 3 gives the description of all packages, classes and interfaces used in CompOrch.

Table 3. Packages, classes, interfaces used in CompOrch

S. No.	Orchestrator Name	Class Name	Interface	Package name	Packages used
1	Comput-ational-Orches-trator	Computational-Orchestrator (super class)		Orches-trator	orchestrator.inverse, orchestrator.strasen, orchestrator.RoboRMI, orchestrator.ControlRobo,java.rmi, java.rmi.registry, java.util,java.io,java.text.
2	Inverse-Orchestra-tor and Inverse-Orchestra-torE	RMIClientMath, RMIServer, RMIServer_2, RMIServer_3, RmiServerCo.	Comatrix-Interface, Method-Impl.	Inverse	java.rmi,java.rmi.registry, java.util,java.io,java.text, orchestrator,Jama.LUDecompositionJama.Matrix,java.net,
3	Math-Orchestra-tor	StrasenClient, StrasenServer1, StrasenServer2, StrasenServer3, StrasenServer4, StrasenServer5, StrasenServer6, StrasenServer7,	Strasen-Interface1, Strasen-Interface2, Strasen-Interface3, Strasen-Interface4, Strasen-Interface5, Strasen-Interface6, Strasen-Interface7.	Strasen	java.io,java.util,java.rmi,java.rmi.registry,orchestrator,
4	Robo-Orchestra-tor	RoboClient, RoboServer, RoboServer2, RoboServer3	Robo-Interface1, Robo-Interface2, Robo-Interface3,	Robo-RMI	java.rmi,java.rmi.registry,java.util,java.io,java.text,orchestrator, gnu.io,java.net.
5	Control-Orchestra-tor	ControlClient, ControlServer1, ControlServer2	Control-Interface1, Control-Interface2	Control-Robo	java.rmi,java.rmi.registry,java.util,java.io,java.text,orchestrator gnu.io, java.net.

7. FEATURES OF CompOrch ARCHITECTURE

Various Orchestrators are available commercially. These are designed for specific applications. ePolicy security Orchestrator [12] orchestrates the security layer. IBM smart cloud Orchestrator [13] orchestrates the computing nodes at clouds. But a composite Orchestrator for the mathematical applications, robots and control systems is not available. Proposed CompOrch is a common architecture of Orchestrator. It is the super class for the orchestration of matrix operations, robots and control systems. It extends to the Orchestrators orchestrating the matrix operations, moving robots and collision detection in moving robots.

There are numerous advantages of a super class for the composite architecture: provides single interface for the multiple type of applications, extended Orchestrator can also work independently, services of extended Orchestrator can also work independently, provides service reusability, provides fault tolerance, Easy to use mechanism, provides high level of abstraction, provides hierarchal structure for Orchestrators and creates objects of specified Orchestrator and work with them.

8. CONCLUSION

The paper described a composite architecture of Orchestrator. Different layers in the architecture and class diagram are implemented and described. The features and steps in functioning of CompOrch are described. The CompOrch extends to Orchestrators in different applications describe in chapters four, five and six.

9. REFERENCES

[1] N. Viswanadham, Kameshwaran, "Orchestrating a Network of Activities in the Value Chain," 5th Annual IEEE Conference on Automation Science and Engineering Bangalore, India, pp. 501-506, August 22-25, 2009.

[2] BPEL basics for java handlers, available at http://www.activevos.com/indepth/c_technology/bPELForJavaDevelopers

[3] Kareliotis Christos, Vassilakis Costas, Georgiadis Panayiotis, "Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling," IEEE International Conference on Web Services (ICWS 2007), pp. 751-758, July 2007.

[4] R.Calinescu, ,Grunske Lars Kwiatkowska, Marta Z. Mirandola, Raffaela Tamburrelli, Giordano, "Dynamic QoS Management and Optimization in Service-Based Systems," IEEE Transactions on Software Engineering, Vol. 37, No.3, pp. 387-409,2011.

[5] Beytullah Yildiz, Geoffrey Fox, Shrideep Pallickara, "An Orchestration for Distributed Web Service Handlers," ICIW 2008, Third International Conference on Internet and Web Applications and Services, pp.638-643, 2008.

[6] Shams, Khawaja S, Powell, Mark W. Crockett, Tom M. Norris, Jeffrey S. Rossi, Ryan Soderstrom, "Polyphony : a workflow orchestration framework for Cloud Computing," 2nd International Symposium on Cloud Computing (Cloud 2010), Melbourne, Victoria, Australia, May 17-20, 2010.

[7] Fumio, O., Jun'ichiro, O., Kunikatsu, T., "An Action Framework for Robots based on Distributed Knowledge Base," IEEE/SICE International Symposium on System Integration, pp.77-82, 4 Dec. 2008.

[8] S.G.Akl., "The Design and Analysis of Parallel Algorithms" Prentice-Hall, 1989.

[9] Jarle Berntsen, "Communication efficient matrix multiplication on hypercube," Journal on Parallel Computing, Vol. 12, No. 3, pp. 335-342. 1989.

[10] D.P. Bertsekas, J.N. Tsitsilk, "Parallel and Distributed Computation: Numerical Methods," Prentice Hall, Englewood, Cliffis, 1989.

[11] Wang, Fei-Yue, Liu, Derong (Eds.), "Networked Control Systems Theory and Applications," Springer publication, 2008.

[12] McAfee ePolicy Orchestrator (ePO), Available at: http://www.mcafee.com/in/products/epolicy-orchestrator.aspx

[13] IBM Cloud Orchestrator, Available at: http://www3.ibm.com/software/products/en/smartcloudorchestrator