DOI: http://dx.doi.org/10.26483/ijarcs.v10i4.6447

ISSN No. 0976-5697

Volume 10, No. 4, July – August 2019

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

DESIGNING A TASK ALLOCATOR FRAMEWORK FOR DISTRIBUTED COMPUTING

Mrityunjay Chaubey
DST-CIMS, Institute of science
Banaras Hindu University
Varanasi, India

Manjari Gupta
DST-CIMS, Institute of science
Banaras Hindu University
Varanasi, India

Abstract: Software Frameworks attempt to capture and implement a software system architecture that is reusable. A frameworks, thus, is a semicode that needs to be customized for a particular reuse. The problem of finding an optimal task allocation in distributed computing system (DCS) is an NP-hard. There are various task allocation algorithms and hence a Task Allocator may implement any of them. Any Task Allocator, hence will have many portions that can be reused to define and implement a Task Allocator. In distributed system a Task Allocation mechanism may be replaced by a new one if a standardized definition of a reusable system architecture for this purpose is available. This work attempts at formalizing a system architecture of a Task Allocator by proposing a framework for the purpose. Here we start the design methodology for OO software and identify the various parts of the software system architecture for task allocation. This effort finally results into a semicode framework. The interesting conclusions include "Identification of that code portion of the semicode framework that does not change when reused", "Characteristics of the code portion that need customization" and the nature of framework definitions that need to be coded at the time of reuse. In this work OO design of various activities of task allocation process has been carried out as per the OO design methodology. To be objects have been identified the dynamic and functional modeling along with identification use cases, corresponding scenarios and data flow diagrams.

Keywords: Distributed system, Object oriented framework , Design, UML, Task Allocation.

I. INTRODUCTION

Today there is a need of techniques and tools to support developers to develop complex software in order to improve current situation. Researchers, in the software engineering fields, have always been striving to find ways and means to develop high quality products at low cost. Software reuse promises substantive reduction in cost and improved quality if seriously planned and practiced. There are certain technical and non-technical reasons that impede reuse [11]. The reuse of software design experiences in form of design patterns and frameworks can make the life of developers easy. It can promise improved maintainability as well. A DCS consists of multiple processing nodes and various cooperating tasks, of any jobs, are distributed across there nodes. Individual nodes schedule the tasks allocated to them. Various tasks of a job are allocated to processing nodes in such a fashion so that execution characteristics of the jobs such as throughput, turn-around time etc are improved. Consequently a DCS requires a ask Allocator that keeps performing task allocation business for all the incoming Tasks. These Task allocators implementations based on corresponding task allocation algorithms [1-6]. There are various task allocation algorithms and hence a Task Allocator may implement any of them. Any Task Allocator, hence will have many portions that can be reused to define and implement a Task Allocator. In distributed system a Task Allocation mechanism may be replaced by a new one if a standardized definition of a reusable system architecture for this purpose is available. This work attempts at formalizing a system architecture of a Task Allocator by proposing a framework for the purpose. Designers of a DCS may required to do task allocation as per new algorithms that may have not been there earlier or in a new situation a new allocator need to be designed as per some other existing algorithm. In such a situation designing and implementing a new allocator would require redoing similar things and undergoing the same design experiences as earlier. As it is obvious most of the things in all Task Allocators will be similar if not always identical. If we can represent that design of a Task Allocator in form of a semicode framework a good deal of reuse of the same would take place this work attempts to achieve the same. An object-oriented framework is a set of collaborating object classes that embody an abstract design to provide solutions for a family of related problems; we have carried out the same for the task allocation.

In the following subsections we have described what an object-oriented framework, methodology we used to develop the framework to appear in this paper and the problem of task allocation. In section 2 we have carried out the design for task allocation in DCS that leads to the framework proposed herein. The concluding section 3 summaries the idea and its usefulness.

II. WHAT IS A FRAMEWORK?

Framework is one of the object-oriented reuse techniques that promise highest degree of reuse among all others. An object-oriented framework is a set of collaborating object classes that embody an abstract design to provide solutions for a family of related problems. The framework typically consists of a mixture of abstract and concrete classes. The abstract classes usually reside in the framework, while the concrete classes reside in the application. A framework, then, is a semicomplete application that contains certain fixed

© 2015-19, IJARCS All Rights Reserved

aspects common to all applications in the problem domain, along with certain variable aspects unique to each application generated from it. According to Ralph Johnson "A frameworks is a reusable design of all or part of a system that is represented by a set of abstract classes and the way there instances interact". Frameworks can be easily refined, reused, customized and extended. Another common definition is "A framework is the skeleton of an application that can be customized by an application developer" [4]. Thus by developing a framework for a particular domain large amount of time and effort can be saved. Since a framework describes the whole architecture of an application and several applications are developed by instantiating this framework its quality must be quite high. The run-time architecture of a framework is characterized by an ``inversion of control." Inversion of control allows the framework (rather than each application) to determine which set of application-specific methods to invoke in response to external events (such as window messages arriving from end-users or packets arriving on communication ports) [8]. This work aims at proposing such a "Framework" for the task allocation. The proposed framework is white-box (based on OO concepts) and vertical (only for the domain of task allocation). The variable aspects (called hot spots), which is here the actual task allocation algorithm, would depend on the different instantiation of this framework. Once the framework is defined for the task allocation it can be used to instantiate a task allocation process for different solution approaches.

III. METHODOLOGY

Here, in brief, we are describing the methods used in developing the framework. Although a framework can be developed using procedural design but to get quality framework it must be developed in object-oriented design. Here framework's quality means its modularity, extensibility, composability and other characteristics that come naturally with object orientation as reusable modules and patterns can also be extracted here from. We have used Object-oriented design [10]. In Object-oriented design object modeling, dynamic modeling and functional modeling is carried out to understand and specify the solution to a problem being designed. Object modeling is performed to identify all the classes relevant to the problem to be considered. Dynamic and functional modeling is performed to identify attributes and operations that can clarify what a class is in the context of considered problem.

OOA and OOD may follow some existing design methodology. A methodology uses some notations for classes, objects and their relations. We have followed the UML that has resulted by unification of Booch and Rumbaugh (OMT) methodologies.

The UML is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The UML provides the several diagrams, to represent user model view, structural model view and behavioral model view of the system [9].

IV. TASK ALLOCATION

In this paper, from now on wards, we are using modules for tasks and tasks for jobs. A Distributed Computing System (DCS) comprising networked of heterogeneous processors. Each processor is of different capabilities for example they have their separate local memory, some may be able to do multithreading while others may not be, some have graphics capability while others may not, etc. DCS provides the users access to various resources so that the system maintains access to shared resources to allow computation speedup and improved data availability and reliability. Processors are connected to each other though links (dedicated or shared). A processor may have access to more than one link. Thus processors connection can be represented using processor graph where a node represents a processor and links represent the communication links among processors. Weight on links represents the communication cost of the links. A task to be run on DCS consists of a set of modules (possibly defined by some partitioning activity). Modules of a task may communicate each other and thus a task can be represented by Task Graph that has each module as a node and communication among modules as links. Weight on a link indicates the communication cost between modules connected by this link. Thus the task graph considers the precedence and the Inter Module Communication (IMC) among the modules. Each of the modules comprising the task will execute on one processor and communicate with other modules of the task. Among all the coming modules some modules can be executed at once (whose all the predecessors are executed) while others have to wait until all their predecessors are executed. Hence, the task allocation on a DCS consists of two major steps: Task partitioning and its allocation [2, 3]. Both the problems are NP-Hard. In a typical DCS, it is possible that some processors are assigned more tasks than others. Therefore, it is desirable for the workload in a DCS to be eventually distributed to maximize the CPU utilization and minimize the average response time. A DCS may have some high and low threshold value for the load to distinguish between heavily and lightly loaded processor.

Thus the task allocation problem can be described as "given a distributed computing system made up of n processors (p1, p2,……..Pn) and several tasks (t1, t2,…….tm), each made up of k modules (m1, m2,……mk), 1<=k<=Large Integer, each module may execute on any processor, allocate each module (of all the tasks) to one of each processors such that an objective cost function is minimized subject to constraints imposed by both the systems and application [1]. In other words we can say the problem of task allocation is to map the tasks, represented by tasks graphs, onto the processing nodes such that it takes optimal time to produce the result.

V. THE PROPOSED FRAMEWORK FOR TASK ALLOCATION IN DCS

Before describing the framework we describe the assumptions that we have taken to propose the framework for task allocation. These assumptions are as follows:

1. There may be several tasks coming in DCS.
2. Module precedence relationship is to be considered.
3. Modules of different tasks may be allocated to processors only if the following conditions are satisfied so that there would not be need of task migration:

i. The resource needed by the module is held by the processor for example if the module requires some graphics capability then for allocation of this module only those

© 2015-19, IJARCS All Rights Reserved

processors would be considered that have the graphics capabilities. Further the local memory held by the processor must be greater than the memory required by the module and so on.
ii. After allocating the module the number of modules assigned to the processor doesn't exceed the number that it can handle at a time.
iii. After allocating the module to the processor its high threshold load does not exceed than a fixed Threshold value (that would depend on the method of finding solution).

Now we start designing for the TA framework. As the first step in OO design is to identify the classes relevant to the problem domain we perform the same. We will concern only those objects that must always be applicable whatever task allocation algorithm one uses. By performing a "*grammatical parse*" on task allocation process described in section 1.3, we identified the following *problem domain classes*: ***Processor*** to which tasks are to be assigned. ***ProcessorGraph* (*ConnectionMatrix)*** describes the links (direct/ single indirect/ double indirect etc.) of connection paths among the processing nodes in Processor Graph (PG). One may also take ProcessorGraph as an attribute of the DCS (described later) class that would represent the whole distributed computing system. It would depend on the designer how he wants to implement it. Here we are taking it as a separate class. ***TaskPartitioner*** that handles the partitioning of incoming tasks into modules. It implies two objects ***Task*** and **Module.** Each task object has many module objects. Further ***FreeModule*** are those modules whose all the predecessors are allocated and thus next at the time of allocation they would be considered while ***UnFreeModule*** represents those tasks whose at least one predecessor has not been allocated and thus they can not be considered in allocation until there all the predecessors are allocated. **FreeModuleFinder** that would find the free modules among all the modules. ***TaskGraph* (TaskPrecedenceGraph) (*Task Interaction Graph)*** describes the communication among the modules. ***PrQueue*** object that acts as queue to which tasks are filled that are allocated to the processor. For each processor object there is a ProcessorQueue object. It is a matter of preference and opinion whether ProcessorQueue should be an object, or ProcessorQueue should be implemented as a property of Processor class. Here we are taking it as a separate class. If required one can combine take it as an attribute of Processor class. ***PrQUpdater*** is an object that updates the processor queue of each processor that is assigns inserts tasks to processor queues (resulted by concerned scheduling policy). ***Threshold*** is a class that encapsulates the high threshold and low threshold value for load of each processor. ***ThresholdUpdater*** is another object that updates high and low threshold of each processor after updation of its processor queue. ***MemManager*** is an object that would update memory of processors after allocation of tasks. ***DCS*** is an object that represents the whole system including all the processors and their links. At an abstract level we can include ***Taskallocator*** as a class that would need to be refined depending on the task allocation algorithm used.
Thus 16 classes are identified in the *problem domain*: Processor, PrGraph, TaskPartitioner, Task, Module, FreeModule, UnFreeModule, FreeModuleFinder, TaskGraph, PrQueue, PrQUpdator, Threshold, ThresholdUpdator, MemManager, DCS, Taskallocator.
Now let us look for *properties* and *methods* of each identified class, as said above, to clarify what a class is in the context of the problem. Each ***processor*** would have a processor Id (PrId), Architectural capability (ArchiCapa), a data structure 'Status' associated with every processing node, which has two fields, showing the maximum no. of the modules that can be allocated to this processor and the memory capacity of the processing node. Another attribute of the processor object is ModulesAssinged that is an integer type array and stores the modules that have been assigned to it. To know when the currently executing module to a processor would get completed there is an attribute remaining execution time (CTT) of the task currently been processed by the processor that would be decremented in each clock cycle until it becomes zero that is until the currently executing task is completed. Resource is another attribute that is an array of resources held by the processor. CurrentLoad, AccepingLoad, Utilization (Util), Throughput and ResponseTimes are other properties of the Processor class. **ProcessorGraph** has an attribute *ConnectionMatrix* that describes the IPC cost for the links of connection paths among the processing nodes. **TaskPartitioner** has method PatitionTask that patitions a single tasks into modules. Each **Task** object has a TaskNo and ModuleNo attributes. ModuleNo corresponds to the number of modules each task contains. InterModuleComm (IMC) is an attribute of each object that gives that inter module communication cost for each task. **Module** object has attributes ExeTime, MemReq, ResourcesReq and a boolean attribute Isallocated. ExeTime, MemReq and ResorcesReq give the time, memory and resources requirement for each module object. Isallocated attribute tells whether the module has been allocated to any of the processor or not. **FreeModule** and **UnFreeModule** are two specialized classes of Module class. **FreeModuleFinder** have one method FindFreeModules that would return all the modules that can be allocated atonce by considering precedence relationships among them. **TaskGraph** object has attribute CommuCost (IMCC) (that can be modeled as two dimentional array) that contains information about communication cost among its modules. **ProcessorQueue** may be empty or it may have some tasks. Thus its one attribute is no of tasks in the processor queue (NoTaskInPrQ). There is total execution time of all the tasks waiting in a processor queue (QT). For initializing, updating the processor queue and its total time there are methods Update QT. Get No of Task In PrQ, Increase No Task In PrQ, Decrease No Task In PrQ, Increase Total PrQ Time, Decrease Total PrQ Time is another method that returns total number of tasks in the processor queue.

© 2015-19, IJARCS All Rights Reserved

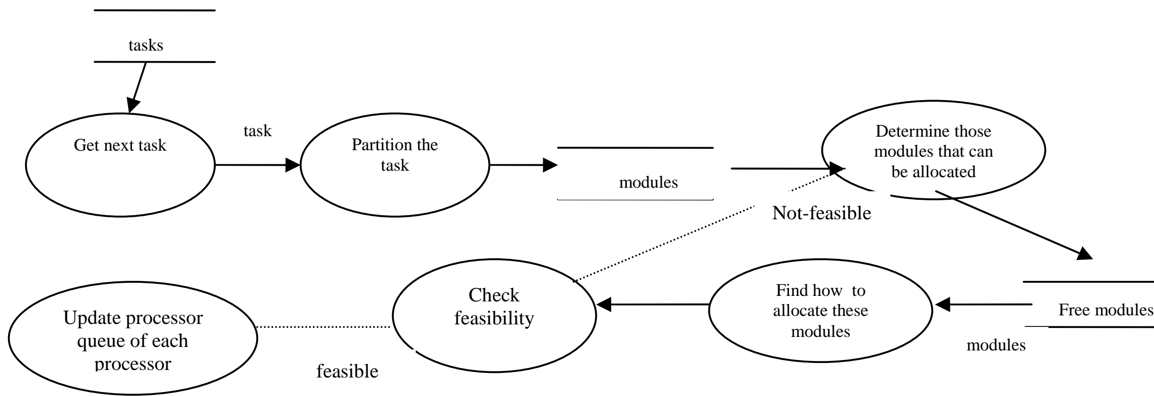

Fig-1 DFD for the task allocation process (for problem domain)

Processor QU pdater class would have method InitializePrQ, UpdatePrQ that would initialize and update the processor queue respectively. Further it would contain AssignTasks as internal methods. **Threshold** class has two attributes high threshold (Th) and Low threshold (Tl). **ThresholdUpdater** would have method UpdateHighTh and UpdateLowTh. **MemManager** has methods ShowMem, FreeMem, AllocateMem. **DCS** object has attributes NoOfPrs, Avgload, AvgUtilization that would show the number of processors in the DCS, overall load and overall utilization of the system. During task allocation one would be interested in reducing the AvgLoad while increasing the AvgUtilization of the system. There always would be one object of PrGraph (described above) for the object DCS. Taskallocator object would have one method AllocateTask that would allocate all the free tasks. The algorithm for this operation would depend on the method to solve the problem of task allocation.

Dynamic Modeling:

The dynamic modeling is used to obtain the behavioral model of the system. the behavioral model indicates how software will respond to external events or stimuli [12]. in the following two sub-sections the dynamic modeling is carried out for the task allocation process. as scenarios should be prepared for both normal and exceptional cases first we give it for normal case and after that for exceptional case.

Scenario:

By developing the scenarios a designer can understand sequence of events that occur in a particular execution of the system [10]. The normal scenario is that the inputs are given and the outputs are produced. The normal scenario for task allocation is as follows:

By assuming that in parallel to the following activities occurred in the system the tasks are coming in the DCS and are partitioned by task partitioner.

Normal Scenario:

1. FreeModuleFinder finds the free modules, that is, that have no predecessor modules (that is can be executed at once).
2. These free modules are given to the TaskAllocator.
3. TaskAllocator generates a solution that suggests which module to assign to which processor.
4. Solution is checked for its feasibility.
4. ProcessorQUpdater updates the processor queue of each processor according to the solution that has the minimum cost.
5. Threshold updater updates the threshold value (high and low threshold) of the processor.

Steps 1 through 5 are executed iteratively until the tasks coming in the DCS stops.

Exceptional Scenario:

There are a number of exceptional scenarios.

I. if a free task is canceled by the user at the time when all the free modules have been considered for allocation but still solution has not been generated:

1. FreeModuleFinder finds the free modules, that is, that have no predecessor modules (that is can be executed at once).
2. These free modules are given to the TaskAllocator.
3. User cancels a task whose some modules are assigned to some processors according to the generated solution.
4. Step 1 and 2 would again execute.
5. And from now events occurs same as in normal scenario.

This scenario hints the need of one more object **User** having the operation **CancelTask** that would delete all the modules (free or un-free both) of the cancelled task. One more operation **IsCancelledAnyFreeModule** on **allocator** that will check whether any free module is canceled or not.

II. If a free task is canceled by the user at the time when solution has been generated by considering modules of these canceled tasks:

1. FreeModuleFinder finds the free modules, that is, that have no predecessor modules (that is can be executed at once).
2. These free modules are given to the TaskAllocator.
3. TaskAllocator generates a solution that suggests which module to assign to which processor.
4. User cancels a task whose some modules are assigned to some processors according to the generated solution.
5. ProcessorQUpdator will count the number of non-empty processor queues.
6. The count is compared by the half of the number of processors.
7. The solution would be discarded. {If this number is less than half of the number of processors.)
8. Steps
7. The remaining modules are assigned to the processors according to the generated solution. {If this number is greater than half of the number of processors.)

© 2015-19, IJARCS All Rights Reserved

8. Steps 1 through 6 are executed until the tasks coming in the DCS stops.
It suggests for adding an operation **Count** on ProcessorQUpdator that would count the number of non-empty processor queues. Further one more operation **Compare** on the same object that will compare this count value to the half of the number of processors.

III. If the task allocation algorithm is not correct or not implemented correctly and generates a solution is not feasible for example if number of modules to any processor, assigned according by the solution, is greater than the empty space in a processor queue of a that processor.

1. FreeModuleFinder finds the free modules, that is, that have no predecessor modules (that is can be executed at once).
2. These free modules are given to the TaskAllocator.
3. TaskAllocator generates a solution that suggests which module to assign to which processor.
4. Solution is checked for its feasibility.
5. One message would be printed giving the reason why the solution is not feasible.
6. ProcessorQUpdater updates the processor queue of each processor according to the solution that has the minimum cost.
7. Threshold updater updates the threshold value (high and low threshold) of the processor.

It indicates to add one more object **FeasibilityChecker** that would check the feasibility of the generated solution. And three operations on this object: **CountNoofModulesAssigned, CountFreeSlotsInPrQ** and **Compare** that would counts the number of modules assigned to each processor according to generated solution, the number of free slots in processor queue of each processor and compares both counts respectively. Further one more object **ErrorGenerator** that would have one method **PrintMessage** should be added that would be executed when the Compare operation results in more number of modules assigned to a processor than the number of free slots in it processor queue.

Functional Modeling:
As said above the functional model is third dimension of object-oriented modeling, here we perform the same for our problem. This dimension helps *in concretize* the application logic and hence helps in making ready the mechanisms of the software to be able to carry out all the computation procedures that the software environment is responsible for. The major transformations and the data flows for task allocation are shown in the following DFD at abstract level:

Object Model:
Having identified all the objects and operations during dynamic (events on objects) and functional modeling (processes in DFD) now the time is to define relationships among all the classes. At this abstract level we come with the relationships, shown in object model (Fig -2), among all the identified classes

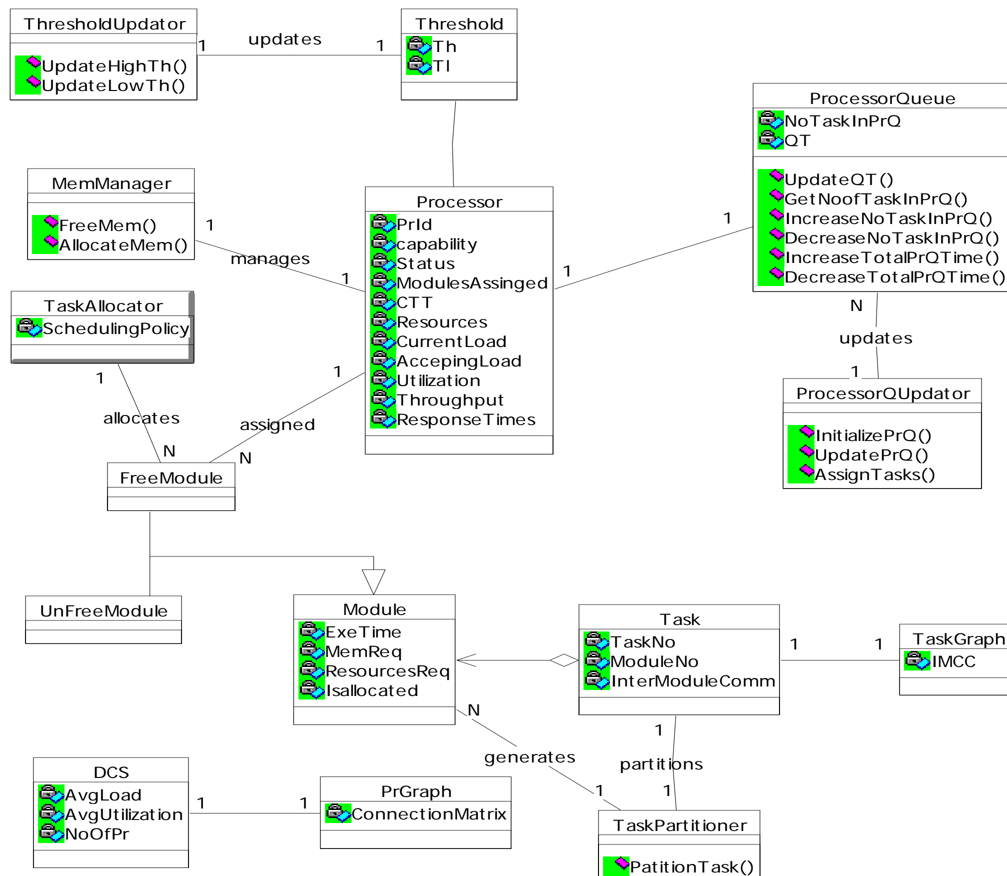

Fig-2 Object model (concerning problem domain) for the task allocation in DCS

© 2015-19, IJARCS All Rights Reserved

Now let us concentrate on the solution domain classes. Although the solution domain classes would depend on the chosen solution method some classes would not directly depend on the method used, that is they would be 'method of solution' invariant, and thus would be common for all the methods used. These classes are as follows:

Solution Domain Classes:

We have got one object TaskAllocator at the time of identifying problem domain classes that is basically a solution domain class. Here, we refine this object to identify classes relevant to solution domain (but common for any approach that may be taken to solve the task allocation problem). These classes are as follows:

Possible*Solution* (resulted on applying any scheduling policy) represent the object that describes the allocation of tasks to the processors. This object has an attribute AllocationCost that represent the cost of the allocation if tasks are allocated according to this solution for example what would be turn around time after allocation of modules according to this solution. **FeasibleSolu** that satifies all the conditions regarding allocation. The classes TaskAllocator, identified during problem domain analysis is required to refine. Thus the refinement of TaskAllocator object is as follows: **PSolutionGen** that would generate the possible solutions. *TaskAllocationAlgo* is one of the most important attribute of the class SolutionGen. **FeasibilityChecker** that would check whether the generated solution is feasible or not. For example whether the module allocated to a processor has the capability that is required to execute the module. **CostEstimator** object that checks the cost of each solution for example the turn around time due to allotment of modules according to this solution. The attributes and operations for solution domain classes would specifically depend on the approach adopting for finding solution. Further other solution domain objects would be specific to the method used to solve the problem.

Dynamic Modeling:

As said above to understand the dynamic behavior of the system one performs the dynamic modeling. Here we are describing the events concerning only with solution domain i.e. at the time of solution generation.

Scenario

A scenario is a sequence of events that occur in a particular execution of the system or with in a group of objects. For the task allocation it is as follows:

1. PSolutionGen gets all the free modules that is have no predecessor modules and thus can be allocated at once.
2. Execution requirements for these modules execution on each processor are checked.
3. Several possible solutions that describe allocation of these modules are generated by PSolutionGen.
4. FeasibiltyChecker checks the feasibility of the solutions by considering several things like resources, threshold, and load of the processors. For example if according to any solution one module is allocated to a processor that has less memory needed by the module the solution would be discarded.
5. CostEstimator estimates the cost of each solution get in the step 5 by considering the task graph (communication cost among modules of each task), processor graph (communication cost among processors) and several other things depending on the used method for solution.
6. ProcessorQUpdater updates the processor queue of each processor according to the solution that has the minimum cost.

Functional Modeling:

To identify all the transformations required from the system one need to perform dynamic modeling. By refining the bubble (process) 'allocate tasks' in the above DFD (Fig-1) we get the following DFD:

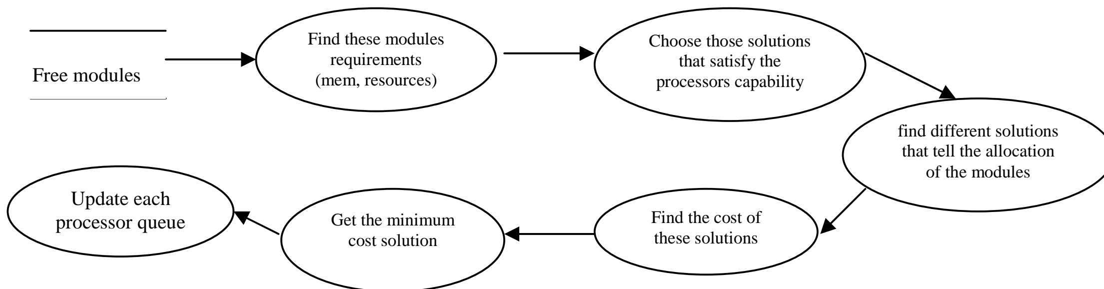

Fig-1 DFD for the task allocation process (for solution domain)

Thus at this abstract level after defining the relationships among these and problem domain classes the obtained object model is as follows:

© 2015-19, IJARCS All Rights Reserved

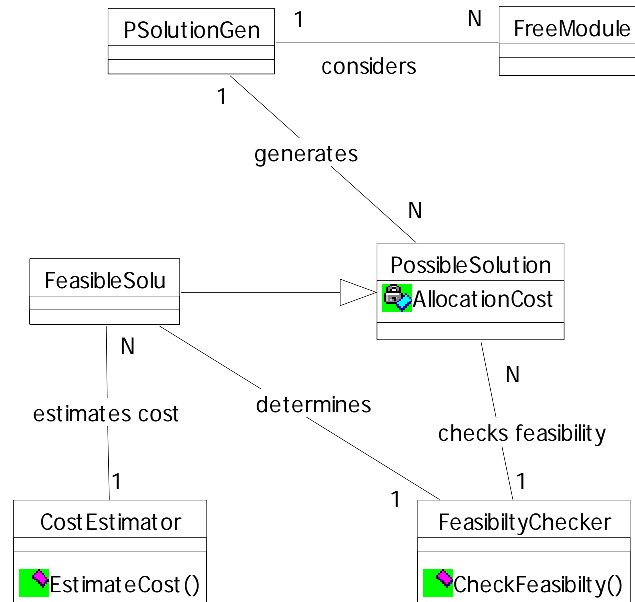

Fig-4 Object model (concerning solution domain) for the task allocation in DCS

VI. CONCLUSION

This paper formalizes the design of task allocation process in DCS in form of a framework. Anybody who wishes to design and implement the task allocation process (applying any solution approach) can make use of this framework. In a way this framework, as any other framework, represents the skeleton as a system architecture that is responsible for the task allocation process. The framework has been developed and designed by object-oriented methodology and hence it formalizes the framework development process. Many of the classes that will always keep appearing in any task allocation process result from this OO design. The specification and the implementation of these parts of the framework need not be rewritten again and again and hence a good deal of reuse would be possible. Researchers and professional organizations may make use of this framework for the purpose.

VII. REFERENCES

1. Kartik S. and Murthy C. S. R., "Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems" IEEE Transactions on Computers, Vol 46, No. 6, Page 719-724, June 1997.
2. Tripathi A.K., Vidyarthi D.P., Mantri A.N., (1996), "A genetic task allocation algorithm for distributed computing system incorporating problem specific knowledge", Int. J. of High Speed Computing, Vol. 8 No. 4, 363-370.
3. Vidyarthi D.P. and Tripathi A.K., "Precedence Constrained Task Allocation in Distributed Computing Systems", International Journal of High Speed Computing, Vol. 8(1), 1996, pp.47-55.
4. Sarker B.K., Tripathi A.K. and Kumar N., "Some observations on Load balancing in Distributed Computing Systems", Proceedings of National Seminar on Applied Systems Engg. and Soft Computing, Agra, 4-5 March, 2000, pp. 167-171.
5. Tripathi A.K., Sarker B.K., Kumar N. and Vidyarthi D.P., "Multiple Task Allocation with Load Consideration in DCS", International Journal of Information and Computer Science, Vol. 3 No. 1, 2000, pp. 36-44.
6. Vidyarthi D.P., Tripathi A.K., "A Fuzzy IMC Cost Reduction Model for Task Allocation In Distributed Computing Systems", fifth International Symposium on Methods and Models in Automation and Robotics, Poland, August 1998, pp 719-721.
7. Gurp J. V. and Bosch J., "Design, Implementation and Evolution of Object Oriented Frameworks: concepts and guidelines", Software-Practice and Experience, pp 277-300, 2001.
8. Fayad M. and Schmidt D. C., "Object-Oriented Application frameworks", Communication of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol 40, No. 10, 1997.
9. Alhir S. S, "Understanding the Unified Modelling Language (UML)", Methods and Tools, published in an International Software Engineering digital newsletter, 1999.
10. Jalote P., "An Integrated Approach to Software Engfineering", Narosa, ISBN 81-7319-271-5, Second Edition.
11. Tripathi A. K. and Gupta M., Some Observations on Reuse Types, Technologies, Practices and Problems, International Journal of Information and Computing Science, Vol.7, No.1, 2004.
12. Pressman. R. S., "Software Engineering a Practitioner's Approach", McGraw Hill International Edition, ISBN 007-124083-7, Sixth Edition.

© 2015-19, IJARCS All Rights Reserved