GCUCE: A Large Scale Grid Computing Environment for Ubiquitous Computing Software Mobility

Dong-Bum Seo, Tae-Dong Lee and Chang-Sung Jeong, *Member, IAENG*

***Abstract*— In this paper, we describes GCUCE (Grid Computing for Ubiquitous Computing Environment) which supports the unified efficient ubiquitous service interacting with Grid service modules using Access Grid computing, which provides the collaborative computing framework which allows the analysis, agreement and discussion among people with a lot of data using Computation Grid Framework (CGF) and Access Grid Framework (AGF). Also, we describe two mobility models: enterprise model and automata model. The former is focused on the service demand aspects, and the latter concentrates on the state transition. Based on those models, we shall show the performance of GCUCE by evaluating several experiments for DOWS (Distributed Object-oriented Wargame Simulation). The two models comprising enterprise model and automata model of software state suggest the formal and mathematical model about software mobility in GCUCE, and provide the overall views and direction of ubiquitous software development.**

Index Terms— **Ubiquitous Computing, Ubiquitous Mobility, Ubiquitous Software Mobility, Access Grid, Computation Grid.**

I. Introduction

The utilization and availability of resources may change computers and network topology when old components are retired, new systems are added, and software and hardware on existing systems are updated and modified [1]-[3]. It is rarely feasible for programmers to rely on standard or default configurations when building applications.

Rather, applications need discover characteristics of their execution environment dynamically, and then either configure aspects of system and application behavior for efficient and robust execution or adapt behavior during program execution. Therefore, an application requirement for discovery, configuration, and adaptation is fundamental to the rapidly changing dynamic environment [4]-[8].

Ubiquitous computing environments [13]-[15], [29]-[30] provide dynamic requirements on applications. Users want to grasp information quickly, navigate representations fluidly, and respond easily. These requirements drive systems toward tight integration where every component knows about all other components in order to simultaneously meet performance expectations and create a composite view that succinctly expresses vital information. These challenges are critical in a young fast-moving field like context-aware computing [9], [10].

A typical system might regularly undergo the addition of new kinds of context sensors, modeled entity types, services, or end-user devices. To reduce fragile coupling among components, software engineering principles argue for anticipating future changes and introducing separation of the identified concerns. The mobility applications will want to support the dynamic execution with successive load and fewer losses [11]-[12].

While ubiquitous computing promises to make many more resources available in any given location, the set of resources that can be used effectively is subject to frequent change - both because the resource pool itself can change dynamically, and because a user may move to a new environment, making some resources available and others not accessible. As we detail later, traditional solutions normally associated with mobile computing [3], [13]-[16] are inadequate to solve this problem either because they are unable to exploit resources as they become available in a user's environment, or because users must pay too high a price to manage those resources.

For the solution to this problem, we insert the concept of Grid computing [4]-[7] into the existing concept of ubiquitous computing [1]-[3]. Many ubiquitous computing systems provide application developers with a powerful framework [8]; however, its design is not intended to support applications requiring either the many resources that are needed to integrate instruments, displays, computational and information managed by diverse organizations, or requiring collaborative environment by audio/video conferencing.

The remainder of the paper is organized as follows: in Section Ⅱ, we describes architecture in GCUCE(Grid Computing for Ubiquitous Computing Environment), and Section Ⅲ explains context reasoning. The Section Ⅳ explains enterprise model in GCUCE, and Section Ⅴ describes automata mobility in GCUCE. Section Ⅵ tests experiments in DOWS on Access Grid and Computation Grid. Finally Section Ⅶ concludes.

This work is partially supported by the Seoul Research and Business Development, Program, Seoul, Korea, MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment), Brain Korea 21 projects in 2007, and Seoul Forum for Industry-University-Research Cooperation, Korea University Grant.

D. B. Seo is with Ph. D course the Department of Information and Communication Engineering in University of Korea, Student Member IEEE (phone: 82-19-673-5144; fax: 82-998-0536; e-mail: treeline@korea.ac.kr).

T. D. Lee is with Ph. D School of Electronics Engineering in Korea University, Korea, he is working at Digital Media, Samsung Electronics (e-mail: leetd@korea.ac.kr).

C. S. Jeong is the associate editor coordinating the review of this paper and approving it for publication, he is a Professor at the department Electronics Engineering in Korea University, Korea (e-mail: csjeong@korea.ac.kr).

II. GCUCE Architecture

The GCUCE is the context-aware ubiquitous computing [9]-[10] environment supporting grid computing [4]-[7], which is composed of three layers as shown in Fig.1: Grid Layer, Context-aware Layer, and Ubiquitous Main Layer.

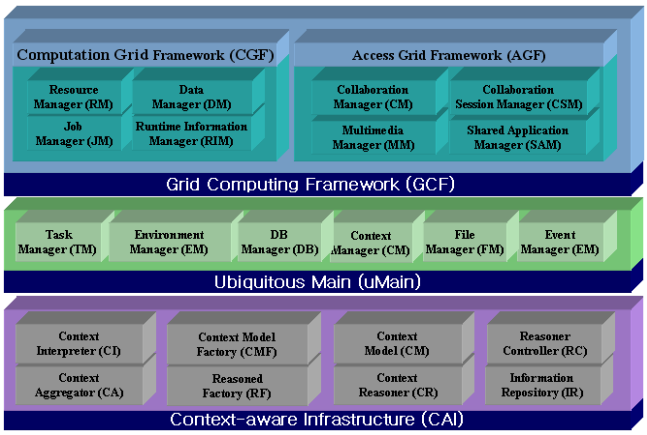


Fig.1. GCUCE Architecture

A. Grid Layer

The Grid Layer [12], [26] is divided into two elements: Computation Grid Framework (CGF), Access Grid Framework (AGF). The CGF provides the functions of grid computing using a Java cog kit, which supports the fault tolerance and high performance computation through resource sharing, monitoring, and allocation. The CGF is composed of four managers: Resource Manager (RM), Data Manager (DM), Job Manager (JM), and Runtime Information Manager (RIM). The RM uses the resource management services offered by Grid, DM provides speed and reliability for files being transferred, JM acts as an agent for the tasks in a job, providing a single entity from which the tasks will request resources, and RIM provides the information to applications or middleware.

The AGF supplies collaboration with audio/video streaming and view of sharing through shared applications on collaborative environments [4]-[8]. The AGF consists of four managers: Collaboration Manager (CM), Collaboration Session Manager (CSM), Shared Application Manager (SAM), and Multimedia Manager (MM). The CM gathers the information about the shared stubs from the venue, CSM provides venue addresses that the user can access, SAM integrates the whole features of the shared applications used in the AccessGrid, and MM controls the base service of AccessGrid like Audio/Video streaming service.

B. Context-aware Layer

The Context-aware Layer [9]-[10] (called Context-Aware Infrastructure (CAI)) has a responsibility of functions, which support the gathering of context information from different sensors and the delivery of appropriate context information to applications.

Also, the CAI supports the context model by ontology methods. The development of formal context models satisfies the need to facilitate context representation, context sharing and semantic interoperability of heterogeneous systems. The CAI provides an abstract context ontology that captures general concepts about basic context, and also provides extensibility for adding domain specific ontology in a hierarchical manner. In CAI, there are eight elements: Context Interpreter (CI), Context Aggregator (CA), Context Model Factory (CMF), Context Model (CM), Reasoner Factory (RF), Context Reasoned (CR), Reasoner Controller (RC), and Information Repository (IR).

The CI gathers contextual information from sensors, manipulates the contextual information, and makes it uniformly available to the platform. The CA processes and aggregates the data through sensor network after context extraction, which provides high-level contexts by interpreting low-level contexts. The CMF defines a context based on concept of specific domain ontology through internal/external providers, and associates a data set with some reasoners to create a CM. The RF creates the specified reasoners, and the CR has the functionality of providing the deduced contexts based on direct contexts, resolving context conflicts and maintaining the consistency of IR, and RC starts and stops the specific CR.

C. Ubiquitous Main Layer

The Ubiquitous Main Layer [11]-[14] (called uMain) is responsible for shielding the user from the underlying complexity and variability through self-tuning environment by mobility and adaptation, which are weak points in CGF and AGF. Whenever the user moves from one place to another, the tasks and devices such as grid authentication, environment variables, video/audio device or large display are automatically set up or executed, keeping their environment.

The uMain has six managers: Task Manager(TM), Environment Manager (EM), DB Manager, Context Manager, File Manager, and Event Manager (EVM). The TM has something concerned with user's task processes. The EM supports services concerned with making same user's environment in everywhere. Database Manager DBM manages the recording about user information. The CM has a role of detection about user's activities such as entering or leaving to/from the environment. The File Manger (FM) has to take a charge for both remote and local file operations. The EVM is to send and receive messages among them locally.

III. Context Reasoning

The basic concept of our context model is based on ontology [33], [35] which provides a vocabulary for representing and sharing context knowledge in a pervasive computing domain, including machine-interpretable definitions of basic concepts in the domain and relations among them. An ontology-based model for context information allows us to describe contexts semantically in a way, which is independent of programming language, underlying operating system or middleware.

A. Context Ontology

There are several reasons for developing context models based on ontology. The use of context ontology enables computational entities such as agents and services in ubiquitous computing environments to have a common set of concepts about context while interacting with one another.

This is knowledge sharing. Based on ontology, context-aware computing can exploit various existing logic

reasoning mechanisms to deduce high-level, conceptual context from low-level, raw context, and to check and solve inconsistent context knowledge due to imperfect sensing. By knowledge reuse well-defined ontologies of different domains, we can compose large-scale context ontology without starting from scratch.

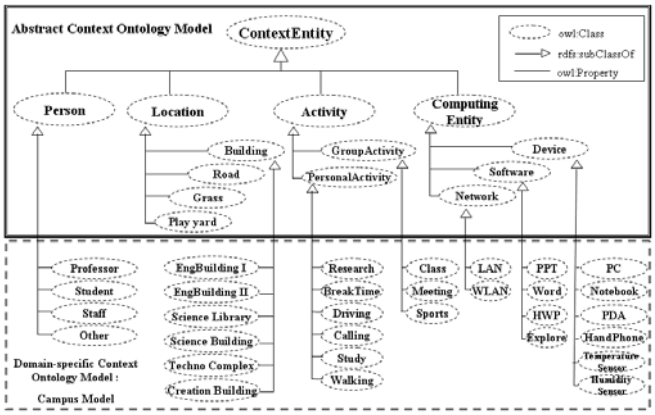


Fig. 2. Context Ontology Campus Model

In Fig. 2 shows the context model, which is divided into two layers: abstract context ontology and campus domain context ontology. The abstract context model is structured around a set of abstract entities, each describing a physical or conceptual object including Person, Location, Activity, and Computational Entity, as well as a set of abstract sub-classes. Each entity is associated with its attributes (represented in owl:DatatypeProperty) and relations with other entities (represented in owl:ObjectProperty). The built-in OWL property owl:subClassOf allows for hierarchically structuring sub-class entities, thus providing extensions to add new concepts that are required in a specific domain. Besides general classes defined in abstract ontology, a number of concrete subclasses are defined to model specific context in a given environment. The campus domain ontology for specific domain model is depicted with inheritance from abstract context model (e.g., the abstract class GroupActivity of campus domain is classified into three sub-classes Class, Meeting, and Sports).

B. Ontology Relationship

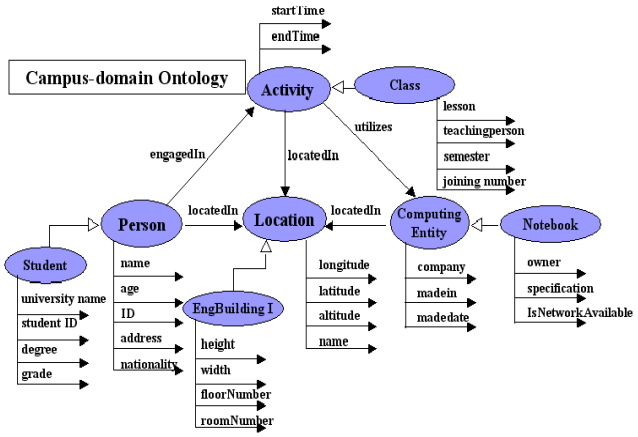


Fig. 3. Simple Context Ontology Relationship

In Fig.3 shows a simple definition of specific ontology for a campus application domain. The Student inherited from Person is engaged in Class with Notebook in EngBuilding I.

Where each concrete values are set such as Student name, Class lesson, Notebook owner, and EngBuildingI roomnumber.

IV. Enterprise Model In GCUCE

The system architecture in uMain has the components: ubiCore and ubiContainer as shown in Fig. 4. The ubiCore is responsible for the application mobility [20]. When a user moves from one place to another; ubiCore provides the automatic movement of computing environment through ubiContainer, which supplies the user information such like IP, user preference, etc.

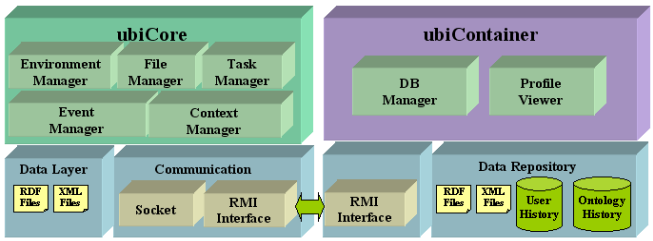


Fig. 4. Architecture of uMain

The communication component in GCUCE uses socket, Java RMI (Remote Method Invocation). The socket used by File Manager for transferring the files. RMI is Java's mechanism for supporting distributed object based computing, which RMI allows client/server based distributed applications to be developed easily because a client application running in a Java virtual machine at one node can invoke objects implemented by a remote Java virtual machine (e.g., a remote service) the same way as local objects.

The RMI mechanism in Java allows distributed application components to communicate via remote object invocations, and RMI mechanism enables applications to exploit distributed object technology rather than low level message passing (e.g., sockets) to meet their communication needs.

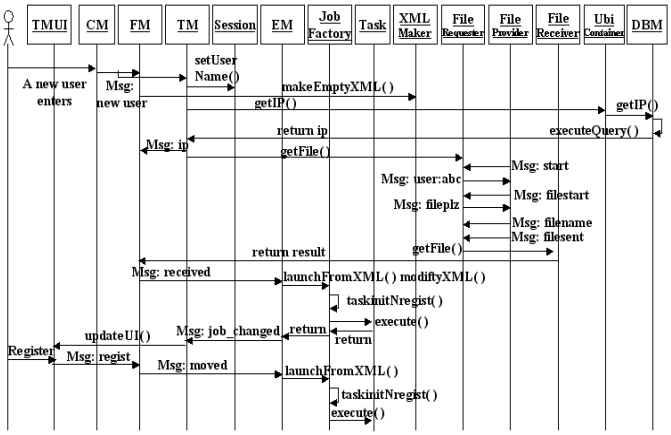


Fig. 5. Sequence Diagram of uMain

In Fig. 5 shows the sequence diagram of uMain. When a new user enters into a new place or device, CM detects the entrance of the user, and TM brings the information related to the user, and FM makes the directory. TM copies the files associated with the user by File Requester. File Requester sends and receives messages to File Provider on remote host, and File Provides sends the files to File Receiver. After the

end of file copies, EM executes the registered tasks through JobFactory, which commands the start of tasks to Task. If a user registers the task through the TMUI(TM user interface), the related files are moved to a user directory and the XML file is updated.

A. Enterprise Model

Let us think about mobility the enterprise model of competitive ubiquitous grid service [8], [9], [22]. Assume a set *G* of *N* grid service providers. That is, $G = \{1, 2, \ldots n, \ldots N\}$. Each provider can be distinguished by three service parameters such as $\{r_n, l_n, m_n\}$, where r_n is the response time of *n*th service provider for unit resource demand from its subscribers and l_n is the loss probability experienced by that service provider, and m_n is the mobility probability accumulated by *n*th service provider.

The selection of these parameters has a significant impact on completion of the job of ubiquitous users within a limited time frame. Due to the competitiveness in between the service providers, the ubiquitous user gets an option to shift from one service provider to another for job completion at the earliest possible time.

So, the *n*th grid service provider experiences a demand d_n, which depends not only on own response time, loss probability, and mobility probability but also on the response time, loss probabilities, and mobility probabilities, offered by its competitors. So, d_n depends upon entire response time vector $r = [r_1, r_2, ..., r_N]$, loss probabilities vector $l = [l_n, l_2, ..., l_N]$, and mobility probabilities vector $m = [m_1, m_2, ..., m_N]$.

The strategy of each grid service provider will be always to provide a response time, loss probability, mobility probability which are in between the maximum and minimum values offered by all of its competitors. Then, the strategy space, S_n of *n*th grid service provider, as in

$$S_n = \left\{ \begin{array}{l} (r_n, l_n, m_n) : 0 \le r_{\min} \le r_n \le r_{\max}; \\ 0 \le l_{\min} \le l_n \le l_{\max}; 0 \le m_{\min} \le m_n \le m_{\max} \end{array} \right\}.$$

We assume $r_{\min}$ depends on l_n in the sense that if the value of the loss probability increases, then the service provider has to decrease its response time r_n. The upper bound on the response time and loss probability express that after a certain value, the demand will be zero.

B. Demand Model

For a particular grid service provider, the demand d_n for its services decreases as its response time r_n increases; on the other hand, it increases with the increase of its competitor response time r_m, for $m \neq n$. The analogous relationship holds for loss probabilities, but then, d_n increases with decrease of l_n and increase of l_m, for $m \neq n$. The reverse relationship holds for mobility probabilities, but then, d_n increases with increase of m_n and decrease of m_m, for $m \neq n$. We now consider the case where the demand function (d_n) is linear in all QoS parameters [12]. That is,

$$d_n(r,l,m) = \left\{ \begin{array}{l} \sum_{m \subseteq G, m \neq n} \alpha_{nm} r_m - \theta_n r_n + \\ \sum_{m \subseteq G, m \neq n} \beta_{nm} l_m - \delta_n l_n - \\ \sum_{m \subseteq G, m \neq n} \chi_{nm} m_m + \eta_n m_n \end{array} \right\} \quad (1)$$

where $\alpha_{nm}, \theta_n, \beta_{nm}, \delta_n, \chi_{nm}, \eta_n$ are constants, and m_n is mobility value at *n*th grid service provider. Here we make some minimal assumptions regarding the demand function.

$$\left[\begin{array}{l} \frac{\partial d_n(r,l,m)}{\partial \gamma_n} \le 0, \frac{\partial d_n(r,l,m)}{\partial l_n} \le 0, \frac{\partial d_n(r,l,m)}{\partial m_n} \ge 0; \\ \frac{\partial d_n(r,l,m)}{\partial \gamma_m} \ge 0, \frac{\partial d_n(r,l,m)}{\partial l_m} \ge 0, \frac{\partial d_n(r,l,m)}{\partial m_m} \le 0; \\ \forall m \neq n \end{array} \right]$$

This results in a decrease of own demand while increasing those of its competitors, if a service provider increases response time (loss probability), assumption the demand d_n is non-negative over the strategy space (i.e., response time and loss probability are decreasing) and negative over the non-strategy space (i.e. response time and loss probability are increasing). Now each service provider n will charge a cost, n_c, per unit of the demand provided to the ubiquitous user. Then the *gain* earned by the *n*th service provider is

$$G_n = c_n \times d_n(r,l,m) = c_n \times d_n \quad (2)$$

We define that ε is the ratio of proportionate change in quantity demanded to proportionate change in cost

$$\varepsilon = -\frac{(\partial d_n)/d_n}{(\partial c_n)/c_n} = -\left(\frac{\partial d_n}{\partial c_n}\right)\left(\frac{c_n}{d_n}\right) \quad (3)$$

Since the revenue of *n*th service provider is given by $G_n = c_n \times d_n$, then taking the partial derivative of both sides we get,

$$\left[\begin{array}{l} \frac{\partial G_n}{\partial c_n} = d_n + c_n \times \frac{\partial d_n}{\partial c_n} \\ \frac{\partial G_n / \partial c_n}{G_n} = \frac{(d_n + c_n \times \partial d_n / \partial c_n)}{G_n} \end{array} \right],$$

$$\left[\frac{\partial G_n / \partial c_n}{G_n} = \frac{1-\varepsilon}{c_n} = -\left(\frac{\varepsilon - 1}{c_n}\right) \right] \quad (4)$$

Now integrating Equation (2) and considering the initial value to demand as k, and then we get,

$$d_n = \frac{k}{c_n^{\varepsilon}} \Rightarrow c_n = \left(\frac{k}{d_n}\right)^{1/\varepsilon} \quad (k \text{ is initial value}) \qquad (5)$$

Equation (5) represents a more generalized demand function by incorporating constant price elasticity model, which is more sensible and appropriate for our scenarios.

C. Gain maximization

The revenue maximization problem is

$$\max_{(c_n, d_n)} G_n = c_n \times d_n(r, l, m) \qquad (6)$$

Subject to the following constraints:

$$c_n \geq 0, \forall n \in N \qquad (7)$$

$$\left\{ \begin{array}{l} \sum_{m \subseteq G, m \neq n} \alpha_{nm} r_m - \theta_n r_n + \sum_{m \subseteq G, m \neq n} \beta_{nm} l_m - \delta_n l_n - \\ \sum_{m \subseteq G, m \neq n} \chi_{nm} m_m + \eta_n m_n \leq d_n(r, l, m), \forall n \in N \end{array} \right\} \qquad (8)$$

In the above formulation, the cost and demand variable $d_n(r, l, m)$ are both present. We will find it convenient to replace the price variable by Equation 6 and retain only the demand variables. The optimum price may be recovered from the demands in the solution. Transforming the objective function gives: max

$$\begin{aligned} \max_{(c_n, d_n)} G_n &= \left(\frac{k}{d_n(r, l, m)}\right)^{1/\varepsilon} \times d_n(r, l, m) \\ &= k^{1/\varepsilon} \times d_n(r, l, m)^{\frac{\varepsilon - 1}{\varepsilon}} \end{aligned} \qquad (9)$$

D. Lagrange multipliers

1) Introduction

In mathematical optimization problems, the method of Lagrange multipliers [34], [39], [45] named after Joseph Louis Lagrange, is a method for finding the extrema of a function of several variables subject to one or more constraints; it is the basic tool in nonlinear constrained optimization. Simply put, the technique is able to determine where on a particular set of points (such as a circle, sphere, or plane) a particular function is the smallest (or largest).

The RMI mechanism in Java allows distributed application more formally; Lagrange multipliers compute the stationary points of the constrained function. By Fermat's theorem, extrema occur either at these points, or on the boundary, or at points where the function is not differentiable. It reduces finding stationary points of a constrained function in n variables with k constraints to finding stationary points of an unconstrained function in $n + k$ variables.

The method introduces a new unknown scalar variable (called the Lagrange multiplier) for each constraint, and defines a new function in terms of the original function, the constraints, and the Lagrange multipliers. Consider a two-dimensional case. Suppose we have a function, $f(x, y)$, to maximize, subject to the constraint $g(x, y) = c$, where c is a constant.

We can visualize contours of f given by

$$f(x, y) = d_n. \qquad (10)$$

For various values of d_n, and the contour of g given by $g(x, y) = c$. Suppose we walk along the contour line with $g = c$. In general the contour lines of f and g may be distinct, so traversing the contour line for $g = c$ could intersect with or cross the contour lines of f. This is equivalent to saying that whilst moving along the contour line for $g = c$ the value of f can vary. Only when the contour line for $g = c$ touches contour lines of f tangentially, we do not increase or decrease the value of f that is, when the contour lines touch but do not cross.

This occurs exactly when the tangential component of the total derivative vanishes: $df = 0$ which is at the constrained stationary points of f (which include the constrained local extrema, assuming f is differentiable). Computationally, this is when the gradient of f is normal to the constraint(s): when $\nabla f = \lambda \nabla g$ for some scalar λ.

A familiar example can be obtained from weather maps, with their contour lines for temperature and pressure: the constrained extrema will occur where the superposed maps show touching lines. Geometrically we translate the tangency condition to saying that the gradients of f and *g* are parallel vectors at the maximum, since the gradients are always normal to the contour lines.

Thus we want points (x, y) where, and, further, $g(x, y) = c$. To incorporate both these conditions into one equation, we introduce an unknown scalar, λ and solve

$$\nabla_{x,y,\lambda} F(x, y, \lambda) = 0 \qquad (11)$$

$$F(x, y, \lambda) = f(x, y) + \lambda(g(x, y) - c), \qquad (12)$$

$$\nabla_{x,y,\lambda} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial \lambda}\right). \qquad (13)$$

2) Justification

As discussed above, we are looking for stationary points of f seen while traveling on the level set $g(x, y) = c$. This occurs just when the gradient of *f* has no component tangential to the level sets of *g*. This condition is equivalent to

$$\nabla_{x,y} f(x,y) = \lambda \nabla_{x,y} g(x,y) \quad (14)$$

for some λ. Stationary points (x, y, λ) of *F* also satisfy $g(x, y) = c$ as can be seen by considering the derivative with respect to λ.

3) Caveat: extrema versus stationary points

Be aware that the solutions are the stationary points of the Lagrangian F, and are saddle points: they are not necessarily *extrema* of F. F is unbounded: given a point (x, y) that doesn't lie on the constraint, letting $\lambda \to \pm\infty$ makes F arbitrarily large or small. However, under certain stronger assumptions, as we shall see below, the strong Lagrangian principle holds, which states that the maxima of f maximize the Lagrangian globally.

4) A more general formulation: the weak Lagrangian principle

Denote the objective function by $f(x)$ and let the constraints be given by $g_k(x) = 0$, perhaps by moving constants to the left, as in $h_k(x) - c_k = g_k(x)$. The domain of f should be an open set containing all points satisfying the constraints. Furthermore, f and the g_k must have continuous first partial derivatives and the gradients of the g_k must not be zero on the domain.

Now, define the Lagrangian, Λ, as

$$\Lambda(x, y) = f + \sum_k \lambda_k g_{k.} \quad (15)$$

k is an index for variables and functions associated with a particular constraint, k. λ without a subscript indicates the vector with elements λ_k, which are taken to be independent variables.

Observe that both the optimization criteria and constraints $g_k x$ are compactly encoded as stationary points of the Lagrangian:

$$\nabla_x \Lambda = 0 \text{ if and only if } \nabla_x f = -\sum_k \lambda_k \nabla_x g_k,$$

∇_x means to take the gradient only with respect to each element in the vector X, instead of all variables, and

$$\nabla_\lambda \Lambda = 0 \text{ implies } g_k = 0.$$

Collectively, the stationary points of the Lagrangian, $\nabla\Lambda = 0$, give a number of unique equations totaling the length of X plus the length of λ. This often makes it possible to solve for every *x* and λ_k, without inverting the g_k. For this reason, the Lagrange multiplier method can be useful in situations where it is easier to find derivatives of the constraint functions than to invert them. Often the Lagrange multipliers have an interpretation as some salient quantity of interest. To see why this might be the case, observe that:

$$\frac{\partial \Lambda}{\partial g_k} = \lambda_k. \quad (16)$$

So, λ_k is the rate of change of the quantity being optimized as a function of the constraint variable. As examples, in Lagrangian mechanics the equations of motion are derived by finding stationary points of the action, the time integral of the difference between kinetic and potential energy. Thus, the force on a particle due to a scalar potential, $F = -\nabla V$, can be interpreted as a Lagrange multiplier determining the change in action (transfer of potential to kinetic energy) following a variation in the particle's constrained trajectory.

In economics, the optimal profit to a player is calculated subject to a constrained space of actions, where a Lagrange multiplier is the value of relaxing a given constraint. Here we use a Lagrange multiplier λ, associated with the constraint implied by demand satisfaction in (9) to form the Lagrange expression

$$L(d_n(r,l,m),\lambda) = k^{1/\varepsilon} \times d_n(r,l,m)^{\frac{\varepsilon-1}{\varepsilon}} + \lambda \left(d_n(r,l,m) - \left(\begin{array}{l} \sum_{m \subseteq G, m \neq n} \alpha_{nm} r_m - \theta_n r_n + \\ \sum_{m \subseteq G, m \neq n} \beta_{nm} l_m - \delta_n l_n - \\ \sum_{m \subseteq G, m \neq n} \chi_{nm} m_m + \eta_n m_n \end{array} \right) \right) \quad (17)$$

Now the first-order condition for maximization of $L(d_n(r,l),\lambda)$ is found by equating the partial derivative of *L* to zero. Thus,

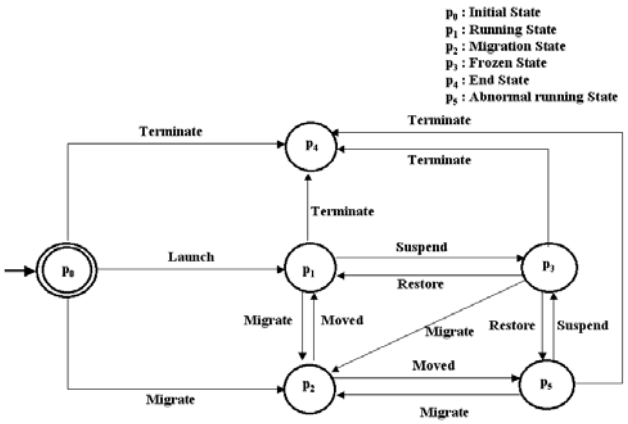
$$\frac{\partial L}{\partial d_n} = 0 \Rightarrow \left[\frac{\varepsilon - 1}{\varepsilon} \left(\frac{k}{d_n(r,l,m)} \right)^{1/\varepsilon} + \lambda = 0 \right]$$
$$\Rightarrow c_n = \left(\frac{\varepsilon - 1}{\varepsilon} \right) \lambda \quad (18)$$

So, the strategy of each grid service provider are will be always to provide a response time, loss probability, mobility probability which are in between the maximum and minimum values offered by all of its competitors.

V. Automata Model for Mobility In GCUCE

Mobile Grid service in grid computing is a new paradigm of Grid service. Grid Mobile Service [19] provides a series of standard interfaces and intelligent mobile code service to computation. It is extension software agent and Grid technologies. In GCUCE, the software mobility is supported.

When sensor detects any signal such as entrance of person, related software is moved into new place, and then executed. The physical happenings can be made into formal automata.



Input / Condition	R	E	F	M	A
00	E	No	E	R	E
01	F	No	R	A	F
10	A	No	A	No	R
11	M	No	M	No	M

Fig. 6. Automata of Software Mobility

Fig. 6 illustrates the software mobility state transition. The initial state is software state beginning the system, running state is software execution state, migration state is the state of transfer from a host to another host, and stop (frozen) state is the termination state of software, and finally end state is the termination state of software.

A. Automata Mobility

1) Automata vocabulary

The basic concepts of symbols, words, alphabets and strings are common to most descriptions of automata. Symbol is an arbitrary datum that has some meaning to or effect on the machine. Symbols are sometimes just called "letters". Word is a finite string formed by the concatenation of a number of symbols. Alphabet is a finite set of symbols. An alphabet is frequently denoted by $\sum$, which is the set of letters in an alphabet.

Language is a set of words, formed by symbols in a given alphabet. May or may not be infinite. Kleene closure a language may be thought of as a subset of all possible words. The set of all possible words may, in turn, be thought of as the set of all possible concatenations of strings. Formally, this set of all possible strings is called a free monoid. It is denoted as $\sum^*$, and the superscript * is called the Kleene stare.

2) Formal description

The 5-tuple represents an automaton, $\langle Q, \sum, \delta, q_0, F \rangle$ where $\sum$ is a finite set of symbols, that we will call the alphabet of the language the automaton accepts. δ is the transition function, that is

$$\delta : Q \times \sum \to Q. \tag{19}$$

For non-deterministic automata, the empty string is an allowed input. q_0 is the start state, that is, the state in which the automaton is when no input has been processed yet (Obviously, $q_0 \in Q$). F is a set of state of Q (i.e. $F \subseteq Q$), called accept states. Given an input letter, one may write the transition function as, using the simple trick of currying, that is, writing $\delta(q, a) = \delta_a(q)$ for all $q \in Q$.

This way, the transition function can be seen in simpler terms: it's just something that "acts" on a state in Q, yielding another state. One may then consider the result of function composition repeatedly applied to the various functions δ_a, δ_b, and so on. Repeated function composition forms monoid. For the transition functions, this monoid is known as the transition monoid, or sometimes the transformation semigroup.

Given a pair of letters, $a, b \in \sum$, one may define a new function, $\hat{\delta}$, by insisting that $\hat{\delta}_{ab} = \delta_a \circ \delta_b$, where denotes function composition. Clearly, this process can be recursively continued, and so one has a recursive definition of a function $\hat{\delta}_\omega$ that is defined for all words $\omega \in \sum^*$, so that one has a map

$$\hat{\delta} : Q \times \sum^* \to Q \tag{20}$$

The construction can also be reversed: given a $\hat{\delta}$, one can reconstruct a δ, and so the two descriptions are equivalent. The triple $\langle Q, \sum, \delta \rangle$ is known as a semiautomaton. Semiautomata underlay automata, in that they are just automata, where one has ignored the starting state and the set of accept states.

The additional notions of a start state and an accept state allow automata to do something the semiautomata cannot: they can recognize a formal language. The language L accepted by a deterministic finite automaton is:

$$L = \left\{ \omega \in \sum^* \mid \hat{\delta}(q_0, \omega) \in F \right\} \tag{21}$$

That is, the language accepted by an automaton is the set of all words ω, over the alphabet $\sum$, that, when given as input to the automaton, will result in its ending in some state from F. Languages that are accepted by automata are called recognizable languages. When the set of states Q is finite, then the automaton is known as a finite state automaton, and the set of all recognizable languages are the regular languages. In fact, there is a strong equivalence: for every regular language, there is a finite state automaton, and vice versa.

As noted above, the set Q need not be finite or countable; it may be taken to be a general topological space, in which case one obtains topological automata. Another possible generalization is the metric automata. In this case, the acceptance of a language is altered: instead of a set inclusion of the final state in $\hat{\delta}(q_0, \omega) \in F$, the acceptance criteria are replaced by a probability, given in terms of the metric

distance between the final state $\hat{\delta}(q_0, \omega)$ and the set F. Certain types of probabilistic automata are metric automata, with the metric being a measure on a probability space.

B. Relationship between Automata and Enterprise Demand Model

Fig. 6 describes automata of software mobility using Table. There is a response time and loss probability parameters are related to running and abnormal running state. The two parameters are measured on that state. The mobility parameter is related to migration state. In the case, we describe the relationship of three parameters. In the running/abnormal running state, the response time and loss probability are main factors, and mobility is important factor when the software is transferred. The automata model is software state transition on ubiquitous computing. The combination of two models provides the system characteristics which software is best on ubiquitous computing environment.

VI. Experiment

In this section, we test two cases [21]-[26]. In the first case, the completion time is measured as the number of task is increased when a user leaves out a lab and enters into another lab. The completion time means the loading and execution of task, and Microsoft PowerPoint is used as task.

In the second case, the test of Shows that the completion time of general tasks with registration of AGF is larger than result. For the test, DOWS on AG is used through AGF as graphic mode. The completion time and frame rate is evaluated. Each evaluation shows the superiority of GCUCE.

A. Task Completion Time

Test of loading and execution of tasks as the number of task goes up in Fig. 7 shows. There is John who is a military analyst. Let us suppose that John is working the tasks at lab A (at Room A in Building A), and he registers his general tasks, which can be, used everywhere. He goes out lab A, and moves lab B (at Room B in Building B). We measure the completion time about loading and execution of task when John arrives at lab B as the number of task is increased.

B. Evaluation with Enterprise Model

Like scenario in Fig. 7, John wants to know the result when DOWS simulation [14], [17]-[18], [35] ends while he moves around building, and continues to do other works such as meeting, eating, etc.

Currently, John is at lab B and registers DOWS and RTI on computation Grid, and executes the DOWS on RTI. After execution, he leaves out the lab, and will go into the meeting room (Room C in Building C) where is off 30 minutes away. The simulation will end 20 minutes later after execution, and is transferred to GCUCE as text result. When John will arrive at Room C in Building C, he will receive and analyze the result file. Access Grid is used. John is at Room C in Building C. Now, he joins into collaboration on AG and registers the SharedDOWS on AGF on GCUCE.

He wants to see SharedDOWS and discusses the simulation states with other military analysts through multimedia conferencing. At the moment after some discussion, John must go to other place (Room D in Building D). When John moves to Room D in Building D, GCUCE makes the computing environment same at Room C in Building C that AG is executed and SharedDOWS is provided to him. For experiments, we used the AGTk2.4 and OpenGL, which are constructed on window-based system, while servers are based on Linux. Our implementation is accomplished on 6 PCs as clients, and 2 servers (Pentium IV 1.7GHz).

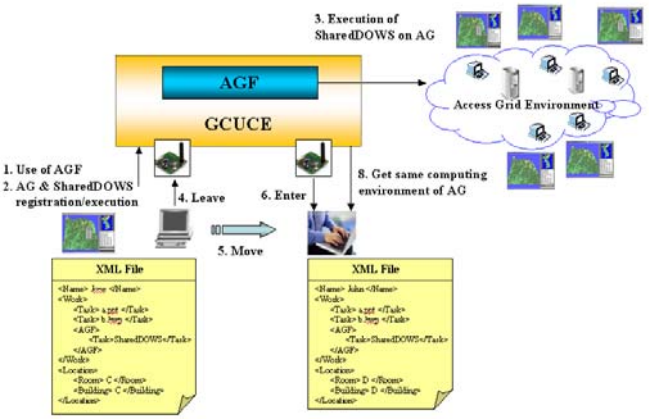


Fig. 7. Test of Shard DOWS on GCUCE

The demand in GCUCE is increasing because the response time in automatic distribution test is shorter, loss rate in dynamic migration test is smaller, and mobility rate in dynamic migration test is higher. GCUCE is superior as the scale of simulation is increasing, although the time consumption of initialization has an effect on the state of small forces. GCUCE can utilize abundant computing power and adapt for various environments, as well as provide convenient user interface. This brings a fast response time, and the demand becomes larger. To verify the dynamic migration service, we execute a second experiment.

In this test, we measured accumulated received packets updated by 600 forces per 30 second. One packet has the size of 100 bytes. In 70 minutes, we intentionally made a failure on one server. The information related to execution like object information is stored periodically, and then the application resigns from the execution. The application sends the stop message to applications before resignation. RM gathers the information, and DM sends the application to a selected host. The application sends the restart message to all applications and receives the data stored before failure. GCUCE can fulfill its mission after the failure, while the original DOWS is halted. The enterprise model shows that responses time and loss probability are decreased, mobility is increased, and the demand of this service is increased.

C. Computation Grid Framework

We use DOWS (Distributed Object-oriented Wargame Simulation) on RTI (RunTime Infrastructure) on Grid [18]. DOWS is an object-oriented simulation system based on a director-actor model, which can be mapped efficiently on object-oriented and distributed simulation. The existing RTIs, the software of HLA (High Level Architecture), do not consider coordinating and managing the resource for distributed simulation to complete the simulation efficiently and effectively. The RTI on Grid is a grid-enabled implementation of RTI solving the problems.

Like the scenario in Fig. 7, John wants to know the result when DOWS simulation ends while he moves around the

building, and continues to do other tasks. Currently, John is at lab B and registers DOWS and RTI on computation Grid, and executes the DOWS on RTI. After execution, he leaves the lab, and will go into the meeting room (Room C in Building C), which is 30 minutes away. The simulation will end 20 minutes after execution, and is transferred to GCUCE as text result.

When John arrives at Room C in Building C, he will receive and analyze the result file. For test, the implementation is accomplished on 4 PCs as clients, and 10 clusters (5: Pentium IV 1.7GHz, 5: Pentium III 1.0GHz Dual) and one 486 computer as servers on a VO. Our experiments are accomplished to confirm key services of CGF. The first experiment is for the automatic distribution service.

We organize the system, which has five servers (we assume that 486 computer is included as server, because of the limitation in local condition), and the GCUCE, which has a VO (Virtual Organization) of 11 servers (10 clusters and 486 PC). Then, we estimated the complete time of simulation as the number of forces increases. As we expected, the resource selection of the GCUCE did not choose the 486 computers.

D. Access Grid Framework

The DOWS system was modified and upgraded into SharedDOWS, which provides the shared view to all DOWS participants. The Access Grid [4], [6] supports shared applications sharing channel such as events channel for collaboration. Having event channel, the client of Access Grid can interact through communicating messages each other. Therefore, we needed shared application so that Shared DOWS combining DOWS into Access Grid is implemented. The Event Channel and the Application Service enable the Venue to provide a mechanism for discovery, coherence, and synchronization among application clients.

1) Architecture of Shared DOWS

When events are appeared, application client transfers their event information through events channel. After receiving the event information of other client, Shared Applications parse the information to apply to their state.

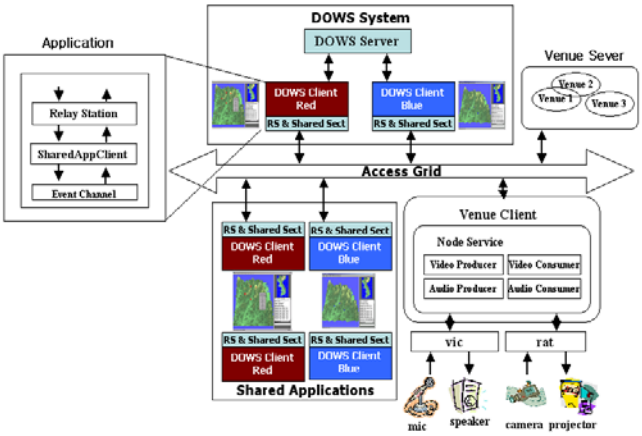


Fig. 8. Architecture of Shared DOWS

Moreover, the AGTk provides developers with a Shared Application Client; implemented in the SharedAppClient class creating shared applications. Shared applications should be run as one of the Venue Client Applications. In order to solve the problem, we also implemented a Relay Station, which has some handlers and message passing method as collaboration method between DOWS and Access Grid [4], [27].

In Fig. 8, we can show both Venue Server and Venue Client. It has to provide people many services such as synchronizing, authorizing and registering users and broadcasting data. In this project, two venues called Blue Team Venue and Red Team Venue are shown in the Venue Client. Entering team venue, you can show a process of simulation of your team using Shared DOWS.

2) Tests on Access Grid

In Fig. 9 shows the comparison between DOWS and SharedDOWS about the simulation completion time and frame rate. The result shows the almost same values, which prove the same system. However, SharedDOWS provides more additional functions than functions of DOWS. The SharedDOWS gives the additional functions of multimedia (audio, video) function and sharing view. The multimedia function makes the military join into the virtual spaces with audio and video, and conference with other military. The sharing view supplies that the military in other host can see the sharing view of applications on another host.

6 clients : Pentium iV 1.7GHz
Windows XP Professional
RAM 1024 M byte
2 servers : Pentium IV 1.7GHz
Linux Hancom 3.0
RAM 512 M byte

Number of forces	50	150	250	350	450	550	650	750	850	950
DOWS(minutes)	2	3.25	4.51	5.5	7.1	9	12.1	15.2	20	25.9
SharedDOWS(minutes)	2	3.26	4.53	5.51	7.1	9.01	12.1	15.2	20	25.9

Number of forces	50	150	250	350	450	550	650	750	850	950
DOWS frame rate	10	10	9	9	8	7	7	7	6	5
SharedDOWS frame rate	10	10	9	8	8	7	7	6	6	5

Fig. 9. Performance Comparison between DOWS and SharedDOWS

If a military want to see the 3D visualization of DOWS, it is possible to see the DOWS visualization through SharedDOWS. While the additional functions are added for SharedDOWS, the evaluation result is almost same as DOWS result.

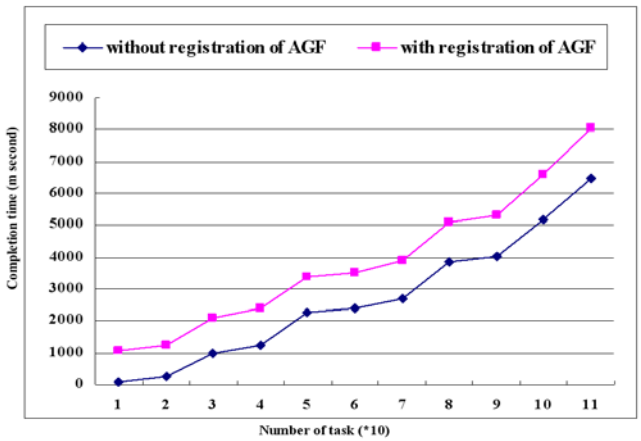


Fig. 10. Comparison of completion time to AGF

Fig. 10 shows that the completion time of general tasks with registration of AGF is larger than result. This means that the loading and execution of AG environment takes one second more, and as the number of general tasks is increased, the loading and execution of AG environment takes more time.

E. Evaluation with AGF

Like scenario in Fig. 7, John wants to know the result when DOWS simulation [24]-[26] ends while he moves around building, and continues to do other works such as meeting, eating, etc.

Currently, John is at lab B and registers DOWS and RTI on computation Grid, and executes the DOWS on RTI. After execution, he leaves out the lab, and will go into the meeting room (Room C in Building C) where is off 30 minutes away. The simulation will end 20 minutes later after execution, and is transferred to GCUCE as text result.

When John will arrive at Room C in Building C, he will receive and analyze the result file. Access Grid is used. John is at Room C in Building C. Now, he joins into collaboration on AG and registers the SharedDOWS on AGF on GCUCE.

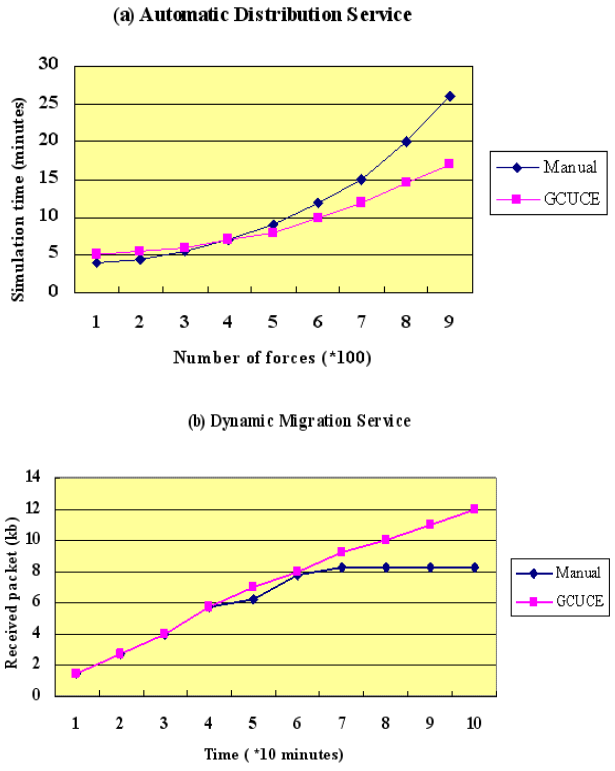


Fig.11. Evaluation and result of GCUCE

As shown in the result of experiment, the demand in GCUCE is increasing because the response time in automatic distribution test is shorter, loss rate in dynamic migration test is smaller, and mobility rate in dynamic migration test is higher. In Fig. 11 (a), GCUCE is superior as the scale of simulation is increasing, although the time consumption of initialization has an effect on the state of small forces. GCUCE can utilize abundant computing power and adapt for various environments, as well as provide convenient user interface. This brings a fast response time, and the demand becomes larger. To verify the dynamic migration service, we execute a second experiment.

In this test, we measured accumulated received packets updated by 600 forces per 30 second. One packet has the size of 100 bytes. In 70 minutes, we intentionally made a failure on one server. The information related to execution like object information is stored periodically, and then the application resigns from the execution. The application sends the stop message to applications before resignation. RM gathers the information, and DM sends the application to a selected host. The application sends the restart message to all applications and receives the data stored before failure.

As shown Fig. 11 (b), GCUCE can fulfill its mission after the failure, while the original DOWS is halted. The enterprise model shows that responses time and loss probabilities are decreased, mobility is increased, and the demand of this service is increased.

VII. Conclusion

We first have described the architecture of GCUCE that is the unified ubiquitous computing environment using grid computing. For collaborative computing, we develop the collaborative computing framework by supporting the real time video and audio stream using high-speed network, it can remote conference.

After explanation of architecture, we suggested the enterprise model of software and automata model of software state. Two models for GCUCE provide the system characteristics, which are designed with the philosophy that supports software mobility among a number of devices for ubiquitous computing makes the system vastly more complex.

The development for solution of the requirements as ubiquitous infrastructure is needed, and we have developed the GCUCE as the core module, which provides the automatic computing environment, and makes applications or services used everywhere.

The enterprise model compares the service aspects among software, and provides the direction of good service. The automata model describes the software state transition on ubiquitous computing. The combination of two models provides the system characteristics which software is best on ubiquitous computing environment.

For the traditional distributed computing model, we have developed two frameworks based on grid computing: Computation Grid Framework (CGF) and Access Grid Framework (AGF). For a large computation application, CGF provides the high performance-computing framework, which processes the real time data with high-speed computation through distributed resources using computation grid. For collaborative computing, AGF supplies the collaborative computing framework, which co-works the analysis, agreement and discussion among people with a lot of data using Access Grid. The framework gives the merits of collaboration with multimedia functions.

The AGF is based on Venue, VenueClient, and Node. We have developed frameworks based on grid computing: Access Grid Framework (AGF) on DOWS system. For a large computation application, AGF supplies the collaborative computing framework, which co-works the analysis, agreement and discussion among the military with a lot of data using Access Grid. The framework gives the merits of collaboration with multimedia functions.

The DOWS system was modified and upgraded into SharedDOWS, which provides the shared view to all DOWS participants. The Access Grid supports shared applications sharing channel such as events channel for collaboration. Having event channel, the client of Access Grid can interact through communicating messages each other.

Therefore, we needed shared application so that Shared DOWS combining DOWS into Access Grid is implemented. Which is designed with the philosophy that supports application mobility among a number of devices for ubiquitous computing makes the system vastly more complex. The development for solution of the requirements as ubiquitous infrastructure is needed, and we have developed the GCUCE as the core modules, which provide the automatic computing environment, and makes applications or services, used everywhere.

For GCUCE superiority, we did several experiments based on two models with military application on AG, including completion time, packet bytes, and frame rate. As a result, the experiments showed the good result of the performance.

REFERENCE

[1] M. Weiser, “Hot Topic Ubiquitous Computing,” Computing publication, 1993, pp. 71–72.

[2] M. Glesner, T. Hollstein, T. Murgan, ”System design challenges in ubiquitous computing environments, ” Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on 6-8 Dec. 2004 Page(s): 11 – 14.

[3] K. Kangas, J. Roning, “Using code mobility to create ubiquitous and active augmented reality in mobile computing,” International Conference on Mobile Computing and Networking archive Proceedings of the 5th annual ACM/IEEE. 1999, Page(s): 48 – 58.

[4] R. Stevens, “Access Grid: Enabling Group Oriented Collaboration on the Grid,” The Grid: Blueprint for a New Computing Infrastructure, C. Kesselman, ed., Morgan Kaufmann, 2003.

[5] I. Foster, C. Kesselman and S. Tuecke, ”The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” International J. Supercomputer Applications, 15(3), 2001.

[6] Access Grid, Available: http://www.accessgrid.org/.

[7] Abowed, G.D, “Software engineering issues for ubiquitous computing**,”** Software Engineering, 1999. Proceedings of the 1999 International Conference on, 16-22 May 1999 Page(s):75 – 84.

[8] R. Harrison, D. Obst, C.W. Chan, “Design of an ontology management framework,” Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on 8-10 Aug. 2005 Page(s): 260 – 266.

[9] T. Gu, H.K Pung, D.Q Zhang, “Toward an OSGi-based infrastructure for context-aware applications,” Pervasive Computing, IEEE Volume 3, Issue 4, Oct- Dec 2004 Page(s): 66 – 74.

[10] W. G Griswold, R. Boyer, S.W Brown, T.M. Truong, “A component architecture for an extensible, highly integrated context-aware computing infrastructure,” Software Engineering, 2003. Proceedings. 25th International Conference, 3-10 May 2003 Page(s): 363 – 372.

[11] T. Kindberg, A. Fox, System software for ubiquitous computing,” Pervasive Computing, IEEE, Volume 1, Issue 1, Jan.-March 2002 Page(s): 70 – 81.

[12] N. Roy, S.K Das, K. Basu, M. Kumar, “Enhancing Availability of Grid Computational Services to Ubiquitous Computing Applications,” Parallel and Distributed Processing Symposium, 2005 Proceedings. 19th IEEE International 04-08 April 2005 Page(s): 92a - 92a.

[13] N. Roy,“Providing Better QoS Assurance to Next Generation Ubiquitous Grid Users”, MS Thesis, University of Teaxs at Arlington, USA, Apr 2004.

[14] IEEE Standard for Modeling and Simulation, “High Level Architecture (HLA) Federate Interface Specification,” IEEE Std 1516.1-2000.

[15] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman - Introduction to Automata Theory, Languages, and Computation (2nd Edition)

[16] Michael Sipser (1997). Introduction to the Theory of Computation. PWS Publishing. ISBN 0-534-94728-X. Part One: Automata and Languages, chapters 1–2, pp.29–122. Section 4.1: Decidable Languages, pp.152–159. Section 5.1: Undecidable Problems from Language Theory, pp.172–183.

[17] James P. Schmeiser, David T. Barnard (1995). Producing a top-down parse order with bottom-up parsing. Elsevier North-Holland.

[18] P. Ghosh, N. Roy, S. K. Das and K. Basu, “A Game Theory based Pricing Strategy for Job Allocation in Mobile Grids”, Proceedings of 18th International Parallel and Distributed Processing Symposium, Snata Fe, New Mexico, Apr 2004.

[19] G. Shang-Fen, Z. Wei, M. Dan and Z. Wen-Li, “Grid Mobile Service: Using Mobile Sofeware Agent Grid Mobile Service,” Machine Learning and Cybernetics, Proceedings of 2004 International Conference on Publication Date: 26-29 Aug. 2004 Volume: 1, page(s): 178- 182.

[20] H. Lamehamedi, S. Zujun, B. Szymanski, E. Deelman, “Simulation of dynamic data replication strategies in Data Grids,” Parallel and Distributed Processing Symposium, 2003. Proceedings. International, 22-26 April 2003 Page(s): 10 pp.

[21] Sun Microsystems: “Documentation Center”, Available: http://www.sun.com.

[22] Microsoft: “Microsoft Technical Resources”, Available: http://www.microsoft.com/net/technical.

[23] Java 2 Platform, Enterprise Edition (J2EE). Available: http://java.sun.com/j2ee.

[24] U.S. Department of Defense (DMSO): “High Level Architecture Interface Specification, Rules, Object Model Template, and Run-Time Infrastructure (RTI) Programmer's Guide Version 1.3 ” http://hla.dmso.mil, 1998.

[25] K. H. Kim and Juqiang Liu: “QoS-driven Resource Management in Real-Time Object Based Distributed Computing Systems”, Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS.01).

[26] I. Foster, C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” Intl J. Supercomputer Applications, 11(2): 115-128, 1997.

[27] M.Jern, S.Palmberg and M.Ranlof, "A robust and easy approach to collaborative visualization" Proceedings of the Sixth International Conference on Information Visualisation (IV’02), IEEE Press, 2002.

[28] G. F. Simmons, Differential Equations. New York: McGraw-Hill, 1972 Page(s): 367.

[29] Abowed, G.D, “Software engineering issues for ubiquitous computing**,”** Software Engineering, 1999. Proceedings of the 1999 International Conference on, 16-22 May 1999 Page(s): 75 – 84.

[30] Tim Kindberg, A. Fox, “System software for ubiquitous computing,” Pervasive Computing, IEEE, Volume 1, Issue 1, Jan.-March 2002 Page(s): 70 – 81.

[31] N. Davies, A. Friday and O. Storz, “Exploring the grid's potential for ubiquitous computing,” Pervasive Computing, IEEE Volume 3, Issue 2, April-June 2004 Page(s): 74 – 75.

[32] G. Arfken, "Lagrange Multipliers." §17.6 in Mathematical Methods for Physicists, 3rd ed. Orlando, FL: Academic Press, 1985 Page(s):945-950.

[33] R. El Azouzi, E. Altman and L.Wynter “Telecommunications Network Equilibrium with Price and Quality-of-Service Characteristics”, Proceedings of the ITC, Berlin, Sept 2003.

[34] J. Tsujii, “Domain ontology and top-level ontology: how can we co-ordinate the two?” Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003 International Conference on 26-29 Oct. 2003 Page(s): 814.

[35] Hsin-Chuan Ho and Chao-Tung Yang Chi-Chung Chang," Building an Elearning Platform by Access Grid and Data Grid Technologies", Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE’04), IEEE Press, 2004.

[36] R. Harrison, D. Obst, C.W. Chan, “Design of an ontology management framework,” Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on 8-10 Aug. 2005 Page(s): 260 – 266.

[37] D. Zwillinger, (Ed.). "Lagrange Multipliers." §5.1.8.1 in CRC Standard Mathematical Tables and Formulae, 31st Ed. Boca Raton, FL: CRC Press, 2003 Page(s): 389-390.