24 – Hour Knowledge Factory: One Instance of Services Computing

Amar Gupta
Thomas R. Brown Chair in Management and Technology
University of Arizona

Rajdeep Bondade
Doctoral Student, University of Arizona

Nathan Denny
University Of Arizona

Abstract—The 24-Hour Knowledge Factory is a service oriented architecture model that goes beyond the conventional global delivery model. It involves round-the-clock operations through the establishment of three or more development teams around the globe. Complex projects are iteratively broken down into simpler modules, with each “composite persona” (CP) being responsible for one such model. Developers around the globe only work during their regular day hours; unfinished work is handed off to the next CP driver. The 24-Hour Knowledge Factory concept brings together business processes and IT services to provide a new services computing solution.

Index Terms—global software development, offshoring, services computing, service oriented architecture

I. INTRODUCTION

“The goal of services computing is to enhance the information architecture adaptability and service-based communication of an organization so it can become more agile along its entire value chain” [27]. The concept of the 24-Hr Knowledge Factory is an instance of services computing whereby businesses can work around the clock, while still permitting all employees to work during daytime in their respective countries.

The 24-Hr knowledge factory concept attempts to impact knowledge-based tasks in the same manner as assembly and manufacturing tasks were impacted by the industrial revolution. The latter revolution led to equipment that was scarce and expensive. Laborers had to work in shifts to help ensure that all equipment was utilized to the maximum possible extent. Over time, this logical concept has been applied to many different domains. Today, the concept of distributed
call centers is based on this fundamental idea. By judiciously placing a cluster of 3 to 4 call centers separated by times zones 6 to 8 hours apart, round-the-clock service can be provided, using employees that work during daytime in their respective countries. In a similar manner, it is now possible for an organization to conduct a growing array of more sophisticated tasks around the clock by placing offices around the globe, enabling such endeavors to be completed in a much smaller timeframe.

The concept of the 24-hour Knowledge Factory involves multiple sites, usually located in three different continents. It is developed around a global delivery model in which members of a global team can work on the same phase of a project, and with each developer working the regular day hours pertaining to their time zone. The 24-hr knowledge factory paradigm is different from the concept of the conventional global development team. In the latter case, the overall project is iteratively segregated into modular units. Each offshore unit is responsible for a distinct subset of such modules, and collaboration between these sites is minimal. This represents horizontal division of the project. The 24-hr knowledge factory, in contrast, suggests a vertical breakdown of tasks, with several global developers working on the same task, sequentially at a temporal level. What this means is that when one developer has worked on a portion of the project, at the end of their workday, this work is handed off to the succeeding offshore site. The developer at this site continues with the same task, with the goal of taking the work-in-progress incrementally closer to completion.
Members of a global development team typically focus on distinct aspects of the same project, based on the horizontal breakdown of tasks; as such, there may not be a high degree of correlation that binds the geographically dispersed members together. These members typically collaborate only on those aspects of the project that are of mutual concern. The 24-Hour Knowledge Factory, on the other hand, requires the global team to be a highly cohesive unit. This is because the 24-Hour Knowledge Factory model requires the offshore sites to communicate often and necessitates frequent transfer of data, information, and knowledge to ensure that each person can perform in an effective manner. This, in turn, places a need to create new types of IT concepts and IT services that are highlighted in this paper. In particular, new mechanisms are needed for individuals to handoff the work-in-progress to their contemporaries located in another continent. The efficiency of handoff is determined by the quality of the knowledge transfer place and the duration it involves. The aim of the 24-Hr Knowledge Factory is to develop IT approaches that facilitate the handoff procedure, so that an employee need to spend only 16 minutes, literally, to learn what his or her colleagues accomplished in the preceding 16 hours. This handoff process varies significantly by the nature of work involved, and the manner in which can be decomposed into mini-tasks.
II. TARGET AUDIENCE FOR THE 24-HOUR KNOWLEDGE FACTORY

There are many different classifications that can be used to categorize work. One such taxonomy organizes work based on the degree of its inherent structure. Work can be highly structured, as in the case of call centers. The other end of the continuum consists of work that is highly unstructured. For example, work performed by world leaders and CEOs of large corporations can be very diverse and unpredictable, not adhering to any rigorous form of structure.

The paradigm of the 24-hour Knowledge factory is appropriate for a growing array of applications that are semi-structured in nature, as highlighted in Figure 2. It is apt for situations where the task can be separated into modules, allowing developers in different locations to closely collaborate on individual modules, with minimal interaction with their peers working on other modules or subtasks. Many applications that are semi-structured in nature allow knowledge to be transferred easily, and at minimal cost.

Note that Figure 2 shows some applications that are related to computers; however, the bulk of the applications are not directly related to computers. In the latter case, the notion of services computing can be visualized as the infrastructure that enables such applications to be performed on computers and for the work-in-progress to be transmitted virtually instantaneously from one collaborating site to another. While the use of the 24-Hour Knowledge Factory paradigm is relevant for the design, development, and implementation of software using geographically dispersed collaborating centers, the bigger challenge and the bigger opportunity is to develop the services computing concepts and infrastructure that can be deployed to create the 24-Hour Knowledge Factories that cater to diverse applications ranging from transportation to healthcare, and from accounting to design of new parts.
III. BUSINESS SERVICES IN THE 24-HOUR KNOWLEDGE FACTORY

A. Offshoring

The 24-Hr Knowledge Factory, due to its fundamental definition, involves “offshoring” as one of its key attributes. Offshoring has become a very important business service employed by global companies to reduce costs, to gain access to wider pool of human resources, and to cut development times.

Many industries follow a development cycle that is sequential in nature. Projects advance by making use of a horizontal sub-division of tasks, such as design, development, testing and verification. In the traditional software development process, these tasks are usually performed in a sequential manner. Code developers usually wait until a particular module of the entire project is complete, before transferring it to the next engineer for testing purposes. However, with the onset of offshoring, one can receive test results on an overnight basis, thereby allowing the developer to build and test portions of the product on an interlaced basis. Earlier, the time...
difference that characterized offshore sites was viewed as a major concern for business. This view has drastically changed, as companies embrace the significant strategic advantages that are inherent to offshoring and related time differences.

B. Handoff

In the case of the 24-hour knowledge factory, handoff of knowledge occurs in a manner very different to that in traditional situations. The prevailing paradigm involves the decomposition of a convoluted task into smaller, less complex modules, with each module being handed to a supervisor or to a peer at the time of completion of that module. Such a horizontal separation of tasks, although requiring the development team to be cohesive as a unit, does not oblige them to interact in a highly coordinated manner due to the high degree of autonomy present in each individual task. In contrast, the 24 hour knowledge factory breaks down tasks in a vertical manner. Multiple developers, distributed around the world, work on the same task, but in a sequential manner; this requires frequent handoff of the work-in-progress, as well as the rationale for the decisions made during the particular shift. Close collaboration between different global teams can improve both the quality and the productivity of the endeavor. This is because through the 24-hour knowledge factory model, the time spent on achieving a task is reduced, with the aid of a vertical breakdown of tasks. This can enhance the utility of the service to the customer, and the profit of the service for the developer [22].

The handoff process is one of the critical success factors (CSF) for the 24-hour knowledge factory concept. Apart from the information sharing aspect itself, there are several other critical issues. For example, employees collaborating in the 24-hr knowledge factory are based in different countries, and therefore cultural differences will exist between these employees. Each society has its own set of tacit rules and customs that need to be considered in order for the
handoff to be effective. Second, some countries limit the type of data that can be exported beyond their shores, based on security, privacy, economic, political, and other considerations. Third, there are issues related to establishment of trust, especially if the different individuals have never met each other on a face-to-face basis. In this paper, we focus primarily on technology and process challenges involved in transcending the temporal and geographic separations between the collaborating centers.

IV. ENABLING TECHNOLOGIES

A number of software vendors have incorporated collaborative software in some of their new products. For example, Microsoft Corp. and IBM have embedded their SharePoint and Quickr technologies respectively into software packages such as Microsoft Word, Excel, and PowerPoint. Such software allows developers to share information without having to locally save any information before it is transmitted via e-mail or a web-based upload.

The Wiki is an example of evolving technology that allows web surfers to add, modify, and remove existing content on their servers. Wikis are outfitted with tools for version control, search engines, and content indices. In a globally dispersed organization, wikis can promote informal knowledge sharing amongst the distributed teams [6]. A wiki allows any person who views a particular document to edit it and to have these changes be instantaneously reflected online.

Distributed authoring and versioning systems, like WEB-DAV, can also facilitate collaboration between offshore teams. Any change that is made to an artifact, such as program code, diagrams, documents, etc. should be reflected at all the other sites that are involved; examples of software that performs this function are RSYNC, CVS, and Subversion [7]. Extreme programming is another relevant concept; it emphasizes collective ownership of code
by all members belonging to the development team [8]. The notion of allowing any developer to make a change to any portion of the project is especially relevant in the context of the 24-hour knowledge factory.

Security technologies, such as digital signatures, encryptions and authentication, are relevant for safeguarding the frequent transmissions of information across continents. Encryption ensures that data is coded, while signatures and authentication procedures are employed to verify genuine senders and receivers.

While the above types of technologies can facilitate the operation of the 24-Hour Knowledge Factory model, the latter motivates the exploration of new concepts that are expressly related to the cyclic and sequential nature of work in such an environment.

V. 24-HOUR KNOWLEDGE FACTORY FROM A SERVICE PERSPECTIVE

An effective service solution should consider three important aspects of their service: the science, the management, and the engineering perspectives of the service system [24]. The science aspect deals with the purpose of the solution to the problem and an understanding of the evolution towards generating the solution. The management aspect deals with how to improve the service system. The engineering aspect deals with how to invent new technologies that will improve the service being delivered. This section attempts to analyze these three aspects of the 24-Hour Knowledge Factory. The discussion below incorporates research performed, under the supervision of the first author, at the University of Arizona and the Massachusetts Institute of Technology.

A. Knowfact

When collaborating on the same phase of a project, each site will make decisions during the
period that it is active. In order to ease the transition of knowledge on such decisions from one site to another, integrated decision support systems are needed to ensure that each worker is able to readily absorb and leverage the information from his or her colleague. A concept demonstration prototype, called KNOWFACT, was developed at MIT to demonstrate the capability to capture and to disseminate knowledge in an actual situation; this involved the design of satellites. In this operational environment, technical decisions relating to the designs of different sub-systems were constantly being made. This, in turn, necessitated frequent re-evaluation by the concerned, geographically dispersed stakeholders. This type of model holds direct relevance to the software engineering domain, where decisions are made on a continuing basis.

Figure 3 depicts the KNOWFACT decision support system. In this block diagram, the Decision Rationale Module (DRM) is responsible for defining the important attributes that define a particular system. This information constitutes the foundation of the utility interview, from which stakeholders can ascertain the activity and progress on the different facets of the project. The importance of this scheme is that it forces the designers to represent the key aspects of the software being developed. The Decision History Module (DHM) accumulates the history of all decision parameters on an evolving basis. The importance of these parameters is then weighted to calculate the values for the attributes that were defined in the DRM. Finally, the calculated values of the different attributes are used in the utility function to determine the overall utility of the system. This utility function is then utilized by the development team to make decisions on the relative importance of attributes for the next stage of the endeavor.

The Decision Rationale Module extracts important information on the objectives of the project and then analyses this information for the stakeholder. This process is conducted on a multi-
attribute level, and allows the stakeholder to interact at this level. Utility and cost-benefit analysis methods are used by the geographically distributed stakeholders for this purpose. The data are analyzed to generate information about the utility and the cost of each possible alternative. This approach allows intelligent decisions to be made, surmounting the problems traditional associated with temporal and geographical separations.

The DHM system is capable of performing automatic capture of important information as decisions are made over time. Information can be viewed in graphical format, history of different states is recorded, integrity checks are performed, and key elements are captured. DHM facilitates the creation of a knowledge repository that stores information on the current state of the activity being performed, the history of the carious states, along with an automatic analysis of system integration integrity. While this functional is mostly automated, the user is often prompted to justify the rationale behind certain decisions when major changes are made to the design.

![Figure 3- KNOWFACT process flow](image)

**B. Composite Personae**

Zhang et al. mention that one of the goals of services computing is to improve the information
architecture adaptability and the service-base communication of an organization [27]. The 24-hour knowledge factory model tries to accomplish this objective through the composite persona (CP) process that is described in this section.

The goal of software engineering is the development of software solutions that can be used to solve complex problems. This is achieved by decomposing the task into many different subtasks, each having its own solution. This decomposition process continues in stages until the sub-problem, and its solution is sufficiently tractable. Each sub-problem may be assigned to a single developer, who owns that artifact through the entire process of coding, testing, and maintenance. This allows the development of black-box solutions, entities whose behavior is well defined and whose inner workings are known, in parts, only to the set of individuals who originally developed these solutions. Ideally, the task as a whole is well defined, allowing for easy categorization of subtasks. If there are only few changes in the setting in which the software is to be distributed, then the amount of lateral communication between the developers can be minimal. However, in most cases, the problem is quite complex and has a high degree of correlation to many of the issues that need to be solved. Conditions external to the project may also change. Thus, changes in the requirements of one sub-system and its behavior will propagate to other systems, creating a ripple of changes that have to be incorporated through all the systems. The result of such an occurrence is a need for considerable cross-communication between all the developers. In the context of a global development team, where sub-tasks are assigned to geographically distributed developers, this can be an imposing challenge.

The use of offshoring has forced developers to collaborate with their contemporaries at different locations. This frequently entails long hours being spent on collaborative tasks; and studies show that developers can expend 70% of their time on such activities [12]. This need for
communication can lead to confusion between teams, which span a range of different languages, time zones, and cultures.

In order to overcome the communication skew and ambiguity between different development teams, caused by differences in culture and linguistics, the 24-Hr Knowledge Factory research group at the University of Arizona has developed the concept of the composite persona. A composite persona (CP) is a highly coordinated micro-team, with properties of both an individual and those of a team of engineers. The CP, to an external observer, seems like a single unit, but internally, it consists of a collection of several individuals. In such a system, each offshore site mirrors its counterpart; i.e. each site has the same number of composite personae. However, this does not imply that an equal number of developers are present at each location, as each site can have individuals that belong to more than one composite persona. During the span of 24 hours, the CP is always active, but with a different driver for each shift of eight hours.

In a scenario that employs composite personae, problems are decomposed into more tractable segments and each module is now assigned to a composite persona, rather than to a single developer. The ownership of the modules and classes is now assigned to a CP, rather than to a single individual. Therefore, for purposes of conflict resolution and discussion, each CP acts as a single entity.

The orthodox global development team employs a horizontal decomposition of tasks, where each task is broken into smaller modules, which are then assigned to different global teams. The concept of composite personae, however, goes one step further and allows tasks to be divided vertically. This vertical division of tasks enforces that all individuals belonging to a particular CP work on the same task, in a time-sequenced manner. Communication plays a major role in determining the success of the composite persona model. There are three different types of
communication that can take place. The first means of communication is handoff. As the succeeding site turns on, and a new driver takes responsibility for the CP, the handoff procedure is necessary for the concerned members of the two teams to discuss appropriate details of tasks that were accomplished and tasks that are still incomplete. For co-located developers, Scrum stand-up meetings can be conducted.

The second type of communication is lateral communication. In this type of communication, the current drivers of two or more CPs, at the active site, are engaged in real-time, synchronous conversation. Since all the concerned persons are at the same location, the ability to engage in real-time conversation is preserved. This type of communication is shown in Figure 4.

![Figure 4- Lateral communication between CP drivers](image)

The third type of communication involves lateral communication along with handoff, with information flowing in both dimensions. Here, for example, a complex conversation may start at one site, but when the site turns off, the drivers hand off to the next site and the conversation resumes. The new drivers use the hand-off time to study the problem and the details of the progress made during the previous shift. Once the problem has been studied, the new drivers resume working on the problem together. Such a communication is shown in Figure 5.
Spohrer emphasizes that for any service to be effective for the customer and profitable for the organization, a very sound understanding of the capabilities and needs of both parties must be realized. However, much of the relevant pieces of tacit knowledge is not often communicated effectively as these pieces are inherently difficult to formalize [23]. For this purpose, it is vital that effective strategies be developed and deployed for knowledge to be represented and transmitted effectively. The next sub-section of this paper focuses on a business strategy that attempts to accomplish this objective.

Overall, large-scale collaboration between different offshore sites around the world is attained with the company having at least one active CP driver working at all times of the day. This approach can provide appropriate degree of agility to the business. This business model can lead to high operational throughout, while still retaining good control and coordination capabilities.

C. CPro

CPro is an agile software process that improves the performance of the CP by incorporating automated algorithms for estimating delivery schedules, reducing software defects, and propagation of knowledge within the CP. It also assigns workloads to the different members of a CP, in a way that maximizes productivity.

CPro incorporates features that facilitate efficient planning during the course of a software project. During the planning phase of project, when it is necessary to determine the schedule to be followed, it is not possible for one person to estimate the entire agenda on behalf of the entire CP. This is because of the vertical decomposition of tasks, which can cause productivity to vary based on the success of the communication process. Further, unforeseen external factors may affect the entire schedule of the project. In order to address this challenge, CPro allows all
developers to give their respective estimates for all the subtasks that have to be performed. A schedule caster then executes a Monte Carlo simulation, based on all the data collected, in order to determine how the project should evolve. The emerging probabilistic schedule can be used by project managers to estimate the delivery dates of various project artifacts.

Figure 6 depicts estimates of different tasks, in a software project, as rated by two different developers.

![Table of estimates for different tasks rated by two developers](image)

**Figure 6- Project complexity estimation**

The schedule caster in CPro then simulates several possible work schedules based on the data obtained. This is shown in Figure 7.

![Simulated work schedules](image)

**Figure 7- Simulated work schedules**

CPro employs a continuous distribution to determine productivity. This distribution is affected by data acquired from the developer’s estimates and from any previous estimates made, including the corresponding factual data. Tasks are treated as continuous entities, as they can be handed-off while still present in an incomplete state, to the next active site. Figure 8 demonstrates a plausible simulation, where incomplete tasks are handed over to the next operational CP.
The schedule caster records previous estimates and the corresponding actual performance from such instances. These data are utilized by CPro to generate high-probability distribution curves, especially ones that will lead to high productivity. For this iterative learning process, two mechanisms were analyzed in detail. The first mechanism used Bayesian probability with joint distributions; and the second one involved case-based reasoning, wherein cases were generated based on prior estimate and actual performance.

For the 24-Hour Knowledge Factory environment, CPro helps to establish a pragmatic definition of services computing, in terms of its ability to assimilate scheduling inputs from all the developers and to provide a cohesive and practical phased schedule for the global development team.

1) CPro Development Process

In the 24-Hour Knowledge Factory environment, the ownership of one or more modules or classes of the aggregate program code is usually endowed upon a CP. CPro disaggregates the project into different phases, such as: planning, design, design review, coding, code review and testing. Each of these phases is atomic; once a task has been accepted by a CP driver, the module must be accomplished in its entirety or dropped altogether. Handoff between two drivers can occur only at a phase boundary. This stipulation may seem to conflict with the vertical breakdown of tasks, which is the cornerstone of the 24-hr Knowledge Factory approach.
However, this is not the case, because each succeeding phase relies heavily on understanding the previous phase; such knowledge is not immediately distributed to developers not belonging to the particular CP.

When a module is assigned to a CP, each member approximates the difficulty of each phase of the task. Qualitative descriptors such as very simple, simple, moderately complex, complex and very complex can be used to describe each phase of the task. Every time a developer makes such an estimate and completes the task, the fact is recorded in the history and CPro adjusts the probability distribution for that developer.

All the qualitative levels of complexity are associated with an offset exponential distribution. Based on the time interval that is available, each developer determines the difficulty of the task. For the purpose of simulation, the values shown in Table 1 can be utilized. The second column provides the developer with a rough measure of the complexity of the task, based on the number of hours required for its completion. After the developer estimates the level of difficulty involved for a particular task, he or she assigns the corresponding complexity level from the first column.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Estimating Boundaries</th>
<th>Simulation Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Simple</td>
<td>(½, 1 ½)</td>
<td>½ + EXP( 2 )</td>
</tr>
<tr>
<td>Simple</td>
<td>(½, 2)</td>
<td>½ + EXP( 1 )</td>
</tr>
<tr>
<td>Moderate</td>
<td>(4, 8)</td>
<td>1 + EXP( 1 )</td>
</tr>
<tr>
<td>Complex</td>
<td>(8, 16)</td>
<td>3/2 + EXP( 1/2 )</td>
</tr>
<tr>
<td>Very Complex</td>
<td>(16, +)</td>
<td>3/2 + EXP( 1/5 )</td>
</tr>
</tbody>
</table>

Two methods were investigated; these are discussed in the following sections.

\(a\) \quad \text{Random work assignment}

In this approach, multiple iterations are performed over the different tasks and each phase of
the tasks, by randomly choosing a developer belonging to a CP. The assumption that all developers are equally committed to the project is considered, and the probability of choosing a developer is the same as the rest. However, in situations where different developers work for different lengths of time, the probability can be appropriately weighted. Once a particular developer is chosen and his or her estimates for the different phases of the task are obtained, a distribution is selected, from which a simulation of the effort required is generated. This effort is recorded before the iteration continues to the next task. This algorithm is shown below in Figure 9.

```plaintext
for each task
  for each phase
    randomly select developer
    retrieve developer's estimate
    simulate effort
    record and tally
```

Figure 9- Algorithm for random work assignment

The random work assignment model is designed to account for unusual cases, where the workload is not uniformly distributed. This scenario occurs, for example, when certain members of the CP are preoccupied with other tasks. This model does not take into consideration the personal preferences of individual developers.

b) Selected Work Assignments

All classes that require coding begin with the planning stage. The driver of the CP starts out by determining the estimates for each phase of the different methods. After this, the developer chooses one of the methods and begins the planning phase. Once this is complete, the driver can move to the planning phase of another method, or can continue working on the design phase of the first method. Thus, at any given time, several open task-phases will exist that the developer
can choose from. Since each phase cannot be segregated further, once a particular phase of a method has commenced, then it must be completed before the end of the developer’s shift. The choice of the task that will be taken on by a developer is influenced by shift timings and the complexity involved. This model considers these decision criteria and is more sophisticated to the random work assignment model.

The selected work assignment model uses a greedy policy of scheduling. When a developer has to make a choice of a task, all open tasks are arranged based on the estimates made by the developer, from very simple to very complex. Usually the easiest task is chosen first. Once a task has been selected, the simulation uses the estimates made by the developer to generate an associated distribution, and simulates the actual effort required for the completion of the task. If the task cannot be accomplished within the available time, and if the developer is unwilling to work beyond the allocated shift time, then that phase is abandoned and remains open for the next developer. It is also possible for the developer to work overtime. The simulation algorithm is capable of locking a particular task, and preventing other developers in the next shift from working on this task. If the task is abandoned, then the lock is released, and the task can be selected by the incoming driver.

2) Simulation results for CPro

In order to simulate the working of CPro, a class called *OrderedCollection* was created; it consisted of two methods, *biset* and *insort*. Estimates for three different developers were created, as shown below. Each model was simulated using 100,000 replications.
Figure 10- Complexity estimates for two methods

\[\begin{array}{c|ccccc}
\text{OrderedCollection} & \text{Plan} & \text{Design} & \text{Design-Review} & \text{Code} & \text{Code-Review} \\
\hline
\text{bisect} & S & C & M & M & S & S \\
\text{insert} & VS & S & S & S & S & S \\
\end{array}\]

\[\text{Graph 1- Probability distribution for random work assignment method}\]

\[\text{Graph 2- Cumulative distribution for random work assignment method}\]

\text{a) Generating a Random Work Assignment Schedule by Simulation}

Using the random work assignment model, the following two graphs were obtained. Graph 1 shows that the time required for completing the task \textit{OrderedCollection} is about 70 hours.
The cumulative distribution curve for the delivery schedule, as measured in calendar days, is depicted Graph 2. This graph shows that with 90% confidence, the project will be completed in five days or less, from the time it commences. It can also be asserted that the project will be completed within four days, with a confidence level of about 75%.

b) Generating a Selected Work Assignment Schedule by Simulation

The two graphs that follow were obtained from data collected from simulations of the selected work assignment model. This model is useful when each member of the CP is completely committed to the current assigned project. The following two graphs illustrate the cumulative distribution curves of the effort and delivery, obtained from the simulation of the selected work assignment model.

Graph 3 shows that with a probability of 90%, the project can be completed in less than 70 hours.

Graph 3- Probability distribution for selected work assignment
Graph 4- Cumulative distribution for selected work assignment

Graph 4 shows delivery estimates that are quite dense. Defect Reduction Strategies CPro also incorporates strategies for defect reduction strategies, as follows:

**Artifact Review:** These are options for design reviews and code reviews, focused on quickly locating errors present in the code. While reading the code, the developer is provided with the logic behind the design, thereby facilitating the process of artifact review.

**Test-driven Development:** This method makes use of unit testing, and involves writing test cases that examine code artifacts in depth. Prior to coding a design, a minimum set of test cases is developed. As the design goes through the cycle of code-test-debug, any new defect that is identified is logged as a new test condition. Eventually, test cases are used to illustrate how the CP, as a whole, has evolved in its ability to understand the problem and the devised solution.

**Heuristic Assignment:** At any given point, there are multiple possibilities of tasks that be performed by the driver of the CP, such as coding, adding a test case, and reviewing the code. CPro utilizes a set of heuristics for recommending a course of action. These heuristics are defined based on the immediate productivity and the long term significance of knowledge sharing. For example, CPro might suggest a developer to review a code, which was not developed by him/her, with the intention of diffusion of knowledge throughout the CP. As a
heuristic, CPro ties to match a particular task-phase to that developer who is likely to be the most productive for the particular task. Tools that incorporate the CPro business model will benefit from its look-ahead scheduling approach for obtaining local optimizations for assigning work to developers.

D. MultiMind

MultiMind is a concept development prototype collaboration tool that incorporates the concept of CPro. MultiMind aims to provide a rich operational environment that employs the 24-hr Knowledge Factory concept, specifically by allowing offshore developers to collaborate using the CP method. MultiMind is more enhanced in comparison to other platforms such as DICE and other collaborative engineering approaches [26]. A high-level architectural view of MultiMind is shown in Figure 11.

![MultiMind Architecture](image)

MultiMind is a tool that guides geographically distributed team members belonging to a CP along the CPro process. Business services concepts and IT technologies, such as Scrum meetings and XP respectively, are also incorporated in MultiMind. The Scrum process, which involves stand-up meetings for the members of the active CP, is automated [10]. This keeps
developer intervention to a minimum, since work summaries are automatically generated by the embedded project management system. On sign-out, this system collects quick reports about the forthcoming progress of the task. During the next shift, upon signing-in, a scrum report is produced based on data obtained from the current state of the project and previously generated scrum reports.

MultiMind is built upon technologies illustrated in the following subsections.

**Lifestream:** Forward communication is critical for the continued evolution of the project. However, there will be instances when the current driver is puzzled as to why a particular decision was taken by the preceding driver. Since the 24-hour knowledge factory does not support real-time communication, some form of proxy of the previous driver is needed. In order to address this requirement, MultiMind utilizes a tool called Lifestream. Lifestream is a monotonically increasing database which logs all objects that are accessed and all events that occurred, with the corresponding time of access or occurrence [10]. Using the Lifestream database, developers can obtain a comprehensive analysis of the state of the project. Lifestream enables developers to elicit the justification behind decisions that were made, when the original developer cannot be contacted in real-time.

In order to further facilitate the driver’s understanding of the current state of the project, the team at the University of Arizona has developed a study tool that has evolved from Activity theory. This is a decision support system in reverse. It reveals all the artifacts that were accessed by the previous worker, which led to that particular decision. This piece of knowledge further clarifies the logic behind a particular decision. Thus, using Lifestream, the developer can travel back in time across the Lifestream and be informed about the sequence of events that took place prior to the decision. The current developer has access to important document and artifacts, web
searches, posted messages and their corresponding replies, and relevant web pages visited, thereby enabling the current driver to replay the sequence of events that led to a particular decision. Figure 12 depicts how Lifestream may appear graphically. Research also is continuing to incorporate intelligent filtering capabilities within the semantic analysis process, in order to remove redundant information and to include only those artifacts and events that are relevant to the decision in question.

![Lifestream Process](image)

**Speech Acts:** Linguists term primary acts used in communication as speech acts. These acts categorize communication into different classifications. The theory is limited to action items within the bounds of a project, and does not dictate communication or impose any limitations. In MultiMind, an embedded discussion tool is present in all project artifact interfaces; this tool utilizes speech act theory to tag all forms of communication according to their purpose. When a new message thread is created by a developer, it may be of the form ‘inquiry’ or ‘inform’, as shown in Figure 13. When a developer responds to a thread, he/she can choose from ‘response’, which answers a question, or ‘inquiry’, in order to pose a counter-question. MultiMind recognizes only three of the five high level speech acts: assertive, directive, and commissive [11].
E. EchoEdit

The simplest manner in which knowledge transfer can take place between two developers is through one-on-one conversation. This is not appropriate for the 24-Hr Knowledge Factory scenario based on time differences and other reasons; hence, proper code documentation becomes very important. Several approaches are available to motivate developers to document their code in a more efficient manner. Literate programming, an IT service concept introduced by Donald Knuth, is a technique that combines language used for documentation with the programming language [13]. Tools such as TeX, Cweb, and METAFONT are based on this principle. The drawback with this solution is that it requires the developer to possess a lucid writing ability, in order to create good documented code. More advanced solutions have been developed by the 24-Hr Knowledge Factory group at the University of Arizona; one of them involves combining code documentation techniques with voice recognition capabilities [4]. This service allows developers to generate audio comments, which are translated into text and are displayed as comments in the code. Audio comments can also be played back to the developer.

The above tool that integrates the code documentation and voice recognition is called EchoEdit; it is based on the composite personae model, and helps to minimize confusion among
geographically dispersed developers. In its current implementation, EchoEdit is a plug-in for the Eclipse environment. This service was targeted towards the IBM Eclipse because the latter is an operational environment that many developers are already comfortable with. EchoEdit allows programmers to choose between using text comments and voice messages; both techniques can be used too. For example, text comments could be used for syntactical or low-level details, while voice messages could be used for addressing high-level ideas. When another programmer enters this environment, voice annotations can be played back by selecting an icon displayed in the margin, similar to that in the PREP environment. As the voice message is played, the relevant block of code is highlighted. The purpose of the voice recognition software is to convert voice notes into text for easy indexing. The 24-Hr Knowledge Factory requires programmers to be responsible for the code after each shift, and to ensure that the code is well documented. EchoEdit supports three user actions: the recording of an audio comment, the playback of the audio file, and the conversion of the audio file into a text comment. An example of the comment, after conversion from audio format, is shown in Figure 14.

Figure 15 shows the design of EchoEdit. The developer interacts with the file in the IDE. When an audio comment needs to be recorded, the user presses the record icon. This action creates an event that is sent to the Record Action Handler. The handler initializes the audio recorder, and then starts listening to the record event. When this event is received, a thread is started to perform the actual recording. Simultaneously, a separate thread is started, which converts voice to text and a dialog box is created, which lets the user known that recording has begun. The audio recorder then waits for an event from the converter thread, after which the converted text format is inserted back into the original file that the user was working on. When a user speaks into the recorder, all data are stored in the memory so that it can be written into an
audio file as soon as the recording is complete.

```java
/**
   * The following method will take in an integer and then return the result after subtracting from that number to the zero.
   * @audioFile HelloWorld.java.1177889903437.wav
   *
   * private int returnDiff (int num)
   * {
   *     int diff = 100;
   *     for (int i = num; i > 0; i--)
   *     {
   *         diff = diff - i;
   *     }
   *     return diff;
   * }
```

Figure 14- Example of EchoEdit

![Block diagram of EchoEdit](image)

Figure 15- Block diagram of EchoEdit

VI. CONCLUSION

“The goal of services computing, however, is the use of information technology (IT) to allow an enterprise to act like a ‘service provider’” [27]. The 24-hour knowledge factory epitomizes the concept of services computing.
The 24-Hour Knowledge Factory is an IT-driven methodology that enables businesses to effectively provide high throughput and highly resilient services to their customers, by establishing effective physical and logical connectivity among members of the development teams. It offers a new business model that brings together collaborating development teams, employing emerging IT as approaches as manifested in MultiMind and EchoEdit, along with business methodologies as exemplified by CPro; these, in turn, are founded on proven methodologies and technologies, such as Scrum, Extreme Programming, and Speech Acts. The successful integration of emerging technologies and business methodologies can greatly enhance the performance of globally distributed teams. By horizontally decomposing tasks between CPs and by enabling a vertical breakdown of the phases within the CP, one can perform sophisticated tasks on a round-the-clock basis, with all individuals working exclusively in the day time of the respective countries. Just as the industrial revolution led to machines and ultimately to round-the-clock manufacturing operations, the 24-Hour Knowledge Factory concept could lead to major innovations in the services computing arena.

REFERENCES


