Distributed Computing Software Building-Blocks for Ubiquitous Computing Societies

K. H. (Kane) Kim

DREAM Lab, Dept. of EECS
University of California
Irvine, CA, 92697 U.S.A.
khkim@uci.edu, http://dream.eng.uci.edu

Abstract: The steady approach of advanced nations toward realization of ubiquitous computing societies has given birth to rapidly growing demands for new-generation distributed computing (DC) applications. Consequently, economic and reliable construction of new-generation DC applications is currently a major issue faced by the software technology research community. What is needed is a new-generation DC software engineering technology which is at least multiple times more effective in constructing new-generation DC applications than the currently practiced technologies are. In particular, this author believes that a new-generation building-block (BB), which is much more advanced than the current-generation DC object that is a small extension of the object model embedded in languages C++, Java, and C#, is needed. Such a BB should enable systematic and economic construction of DC applications that are capable of taking critical actions with 100-microsecond-level or even 1-microsecond-level timing accuracy, fault tolerance, and security enforcement while being easily expandable and taking advantage of all sorts of network connectivity. Some directions considered worth pursuing for finding such BBs are discussed.

Keywords: distributed computing, ubiquitous computing, software engineering, building-block, real time, object, class, component, global time, fault tolerance, security, mobile agent, web service component.

1 Growing needs for new-generation distributed computing software building-blocks

Economically leading countries are evolving steadily toward ubiquitous computing societies (UCSs) and becoming increasingly concerned with the quality of such societies approaching. In such societies, citizens are expected to benefit from various kinds of automated services provided all the time everywhere. Cars, home appliances, pocket appliances, and health aids are expected to become much more intelligent and helpful than they are now. These UCS construction movements have been
largely stimulated by the explosive advances in communication technologies that have been occurring since early 1990's.

Emergence of UCSs means enormous increase in both the number of computing nodes connected together and the number of distributed computing (DC) applications. That in turn means enormous increase in the complexity of DC occurring. Therefore, economic and reliable construction of new-generation DC applications is currently a major issue faced by the software technology research community.

What is needed is a new-generation DC software engineering technology which is at least multiple times more effective in constructing new-generation DC applications than the technologies practiced currently in industry are. Without such advances in the DC software engineering technology, application systems such as next-generation multi-party video-conferencing systems, real-time virtual reality systems, systems of cooperating autonomous ground vehicles, and next-generation secure villages and towns, cannot be constructed with sufficient economy, efficiency, and reliability.

A key part of a new-generation DC software engineering technology should be a new-generation software building-block (BB). The desired types of BBs are those that are significantly more advanced than any types of BBs established before, including the current-generation DC object [OMG02] that is a small extension of the object model embedded in languages C++, Java, and C#. If we are to draw an analogy from the field of building construction, it is easy to see that the BBs used for construction of one-story or two-story homes are hopelessly inadequate for use in constructing 200-story high-rise buildings. Desirable characteristics of the new-generation BBs and some directions worth pursuing for finding such BBs are discussed here.

In Section 2, a brief review of the evolution of software BBs is given first and the desirable characteristics of new-generation DC software BBs are discussed. The latest generation of BBs that have been developed at least in some skeleton forms are classified as the 4.5th-generation (4.5G) BBs. A 5th-generation (5G) BB is defined as a unification of these 4.5G BBs. Then in Section 3, desirable characteristics of 4.5G BBs are discussed in more detail and promising directions for further development and maturing of the 4.5G BBs are presented. An example of a real-time (RT) DC component model, which is one of the 4.5G BBs, is discussed in more depth here. Various challenges in integrating these 4.5G BBs, which is the key to obtaining a desirable 5G BB, are discussed. The paper concludes in Section 4.

2 Desirable characteristics of new-generation distributed computing software building-blocks
In order to make the meaning of a new-generation software BB a concrete one, previously developed BBs and those expected to be established in the next decade are classified as follows.

1st-generation (1G) BB: Subroutine in FORTRAN, procedure in ALGOL and PASCAL, and function in Language C.

2nd-generation (2G) BB: Function in C, etc., with concurrency-control application programming interfaces (APIs).

3rd-generation (3G) BB: Object-class in C++, Java, C#, etc., with concurrency-control APIs (since mid-1980’s)

4th-generation (4G) BB: DC component / object-class (since early 1990’s)

4.5th-generation (4.5G) BB: Real-time (RT) DC component/ object-class (since early 1990’s) [Bol00, Kim97, OMG02], Fault-tolerant (FT) DC component/ object-class (since early 1990’s) [Mos00, OMG04], Secure DC component/ object-class (since mid-1990’s), Mobile agent component/ object-class (since mid-1990’s), Web service component (since late 1990’s) [Boo04, Ses05].

5th-generation (5G) BB: A unification of 4.5-th generation BBs.

4.5G BBs partially address the following desirable characteristics of a DC software BB needed for cost-effective and reliable construction of high-quality UCSs.

1) Many new-generation DC applications which will form an integral part of a high-quality UCS are of reliability-critical nature. In each of such applications, a certain level of the quality-of-service (QoS) must be guaranteed. Otherwise, the general public is not likely to accept such application development as a professional engineering activity. For example, suppose an old single person with fragile health is living alone and her health condition is monitored constantly by various devices on and off her body. When an abnormal condition of her health has been detected, the detecting device tries to send an urgent report to an in-house robot which may attempt a limited emergency care operation as well as to a remote medical decision-maker (e.g., an ambulance dispatcher). The probability of such a report reaching the destination in time in spite of non-negligible types of failures of the computing nodes or communication networks, must be above a certain threshold. The local robot must also perform effectively in spite of some internal failures. Therefore, sufficient fault tolerance capabilities must be built in. This means that the desired BBs must be effective in constructing FT DC application systems. The FT DC components or object-classes established so far fall short of meeting this requirement satisfactorily.
(2) A substantial percentage of new-generation DC applications which will form an integral part of a high-quality UCS are of RT computing types. Those involve actions subject to relatively high-precision timing requirements. Multi-media network based applications, collaborating robots, and sensor network based applications are representative of the exploding RT DC application fields. For example, suppose that some of the functions currently handled by human drivers of cars are to be delegated to the computers controlling the cars. If the control computer of a car collects sensor data from various sources including other cars nearby and detects the danger of possible collision against another car approaching from the right side or the front area, then the control computer must take actions aimed for averting the collision in time. Many such applications have the complexity that cannot be easily conquered by continuous use of low-level RT DC programming technologies involving typically assembly languages or the C language and direct manipulations of low-level communication protocols, thereby forcing the programmers to work with low-level abstractions of various computing parts. This means that the desired BBs must be effective in constructing RT DC application systems. The RT DC components or object-classes established so far are not sufficiently mature yet [Bol00, Kim97, Kim00b, OMG02].

(3) Enormous increase in both the number of computing nodes and their connectivity brings the needs for highly cost-effective security enforcement. Research investments and progresses in this area have been advancing vigorously in recent years. The desired BBs must be effective in constructing secure DC application systems.

(4) Use of desired BBs must lead naturally to system designs which consist of subsystems possessing sufficient degrees of autonomy [Mor86, Kim95]. Such substantially decentralized systems are easy to maintain and expand. Subsystems of such systems may enjoy autonomy in several areas. For example, a subsystem may be designed to spontaneously and periodically take a round of activities without receiving a start signal from another subsystem. It may also read a request from another subsystem at its own convenient time. A subsystem may invoke the services of another subsystem without receiving any guide from other subsystems for doing it. One highly useful direction for loosening the coupling among the subsystems, thereby increasing their autonomous characteristics, is to provide a global time base accessible to all subsystems and let subsystems take coordinated actions on the basis of the global time. This principle of global-time-based coordination of distributed computing actions (TCoDA) was advocated first by Hermann Kopetz more than 20 years ago [Kop85, Kop87, Kop97] but only a small fraction of the potential of TCoDA in the field of DC has been realized. Compared to conventional asynchronous hand-shaking message-based coordination, the TCoDA approach can be multiple times, if not an order-of-magnitude, more efficient and reliable, especially in environments where communication delays are large. The main reason for the slow exploration is the incorrect
perception that most computing system engineers have had of the difficulty of establishing a
global time base which is of sufficiently high precision for various applications of interest to
them. Actually the reality is far better and improving continuously with declining costs of GPS
receivers, etc [Kop97, Kim04b].

In addition, DC application systems composed of desirable new-generation BBs should be easily
expandable and capable of taking advantage of all sorts of network connectivity [Hil02, Fos01,
Kim04a, Kim04c, She01, Sma03]. The aforementioned desirable characteristics are insufficiently
possessed by the 4.5G BBs established so far. A 5G BB, which has not appeared even in a skeleton
form, should have all the characteristics when it is developed.

It is obvious that further development and maturing of 4.5G BBs and establishing a 5G BB
through integration of 4.5G BBs will proceed in parallel for many years to come before the 5G BB
establishment enters the final integration and maturing phase.

3 4.5th-generation software building-blocks

3.1 Evolution of RT DC component models

By the beginning of 1990's, many interesting cases of RT systems became DC systems.
Researchers started seeing DC objects, not processes, as highly promising basic BBs for RT DC
application software. In mid-1990’s, organized industry efforts aimed for establishing some standard
RT DC object programming approaches and language tools started. RT CORBA [OMG02], RT Java
[Bol00], and RT UML [Sel03, Sel07] are the most notable industry efforts. Around year 2000,
Microsoft’s Embedded .Net technology development started. At this time, all these efforts are still at a very early stage and expected to evolve for many years to come.

The levels of abstraction at which the RT DC application programmers using current versions of RT CORBA and RT Java are forced to exercise their logic are not considered to be sufficiently high to induce a quantum jump in the productivity of modern RT DC programmers. Current-generation RT CORBA programmers have to deal with many low-level parameters such as priority numbers, etc. RT Java has not yet advanced to the point where remote method calls with the attachment of deadlines for result returns are allowed. In fact, the current practice in real-time programming in industry is predominantly to use 3G BBs and 2G BBs along with thread programming with priority numbers.

Establishing a practical RT DC programming model which is more abstract than those explored currently in industry and yet does not lower the degree of control over timing precisions of important actions, is considered to be a challenge of critical importance in this decade [Kim04b].

Some academic research efforts are also under way in this area [ISO, WOR].

3.2 TMO: An advanced RT DC component as a 4.5th-generation software building-block

In 1992, the author started an academic research project with the skeleton of a concrete syntactic structure and execution semantics of a high-level RT DC object named the Time-triggered Message-triggered Object (TMO) [Kim93, Kim97, Kim00b, Kim02a, Kim03]. The goal was to relieve the RT DC application designers and programmers of the burden of dealing with low-level programming tools and low-level abstractions of computing and communication environments. The TMO programming and specification scheme has been enhanced in several steps along with supporting tools (kernel, middleware, API, specification, etc) by the author and his collaborators since then.

TMO is reviewed here in relatively greater depth because it is judged to be a good starting point for obtaining a solid industry-strength RT DC component model. It also illustrates a promising direction for searching for a 5G BB.

TMO is a high-level RT DC object model and facilitates a highly abstract programming style without compromising the degree of control over timing precisions of important actions. In terms of raising the level of abstraction dealt with by RT DC application programming, the TMO programming scheme is targeting higher than any other practical RT DC programming model established so far does. The TMO programming scheme and supporting tools have been used in a broad range of basic research projects and also used in an undergraduate course on RT DC programming at UCI for several years (http://dream.eng.uci.edu/eecs123/learn.htm). However, serious efforts for transferring the technology into industry are about to begin only now (http://www.tmoso.org). Feasibility of
establishing TMO programming facilities on the basis of CORBA was also demonstrated a few years ago [Kim02d].

In formulating the TMO, the DC object of which object methods may be called by remote client objects resident in different DC nodes, was taken as the starting point. Figure 1 depicts such a situation. The key question was what would be the “minimal set of new features” that could be added to the DC object to produce an RT DC object. This minimal set of new features must be of such type that could be easily understood by business application programmers and software engineers, not necessary experienced RT programmers. Our judgment was that minimal features were the method completion deadlines and the “wait-until-7pm” features as depicted in Figure 2. The expression “7pm” here implies that the global time base, i.e., the facility allowing a computation-unit to always learn about the current world time with the precision and accuracy sufficient for the given application, is required to support the new RT DC object.

Soon it became apparent that adoption of method completion deadlines in an object dictated the concomitant adoption of the deadlines for arrival of results returned from the remote object methods called by the former object as depicted in Figure 3. In case the deadlines for result arrivals are violated, then it is difficult to guarantee that the relevant method completion deadlines in the client object will be met. In addition, incorporating mechanisms for specifying deadlines for result arrivals are a major step forward in facilitating systematic composition of higher-level services with timeliness assurances out of lower-level services.

Figure 2. Minimal features added to convert the DC object into a real-time DC object

All time references are of global time types.
Thereafter, additional important mechanisms were added to form the TMO. The structure of the TMO is depicted in Figure 4. The TMO is a syntactically simple and natural but semantically major extension of the pervasive object structure [Kim97, Kim00b, Kim02a, Kim02b, Kim03]. It extends the latter into a powerful RT DC software BB with the following characteristics.

(1) The TMO has **time-triggered methods**, also called **spontaneous methods** (SpMs), of which executions are triggered by reaching of the global time at pre-specified time-points, in addition to conventional service methods (SvMs) triggered by requests from client objects. All time references in a TMO are references to **global time** in that their meaning and correctness are unaffected by the location of the TMO. For example, the triggering times may be specified as "for \( t = \text{from} \ 10\text{am} \ \text{to} \ 10:50\text{am} \ \text{every} \ 30\text{min} \ \text{start-during} \ (t, t+5\text{min}) \ \text{finish-by} \ t+10\text{min}". By using SpMs, TCoDA can be easily designed and realized. The new programming power that time-triggered methods bring in has turned out to be enormous. Due to its SpM part, the TMO is an autonomous active DC component.

(2) TMO programmers need to specify action timings in intuitive and easy-to-understand forms only, using mechanisms with strong expressive power. As mentioned earlier, TMO programmers specify a completion deadline for every object method. In the case of a time-triggered method a start time-window is also specified. Once they understand the timing requirements inherent and action timings desired in a given application, they can express their understanding in terms of start-time-windows and completion deadlines without any significant mental effort. They are relieved from the old painful practice of playing threads and their priorities of which a proper manipulation to meet desired start-time-windows and deadlines in DC is almost a magic.
With the TMO programming scheme, designing a remote RT method call takes writing just two statements. Moreover, a remote method call can be accompanied by parameter specifying a “round-trip deadline”, i.e., a deadline for result arrival [Kim00b]. This is a very natural high-level approach to specification of interaction timing requirements. To achieve the effect of a remote RT method call with a deadline imposed for result arrival by playing threads, sockets, and fixed priorities attached to various computation units, requires writing hundreds of statements, in contrast to the two statements in the TMO scheme.

TMOs can use another interaction mode in which messages can be exchanged over logical message channels. Access gates for the logical multicast channels are explicitly specified as data members of involved objects. The channel facility adopted in the TMO scheme is called the Real-time Multicast and Memory-replication Channel (RMMC) [Kim00b], of which an earlier version was called the HU data field channel [Kim95]. The RMMC scheme facilitates RT publisher-subscriber channel in one of the most powerful forms. It supports not only conventional event messages but also state messages based on distributed replicated memory semantics [Kop97].

The TMO scheme has been formulated from the beginning with the objective of enabling design-time guaranteeing of timely actions with significantly reduced efforts than what is required with other practiced RT DC software engineering techniques. The TMO incorporates several rules for execution
of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way [Kim02c].

(6) An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs, which may be produced in a top-down multi-step fashion [Kim97]. In fact, formulated along with the TMO and the associated programming model was a vision of the **TMO Network Development Methodology** (TMONDem) which is a methodology for top-down multi-level design and implementation of both an RT DC application system and its environment simulator [Kim93, Kim97, Kim03].

The designer of each TMO provides a guarantee of timely service capabilities of the object. The designer does so by indicating in the specification of each SvM the *guaranteed execution duration bound* (GEDB) of the SvM, the *guaranteed completion time* (GCT) for every output produced by the SvM, and the GCT for every output produced by each SpM executed on requests from the SvM. The GEDB is a bound on the amount of time spent until the SvM execution is completed after a message triggering the SvM execution activation arrives at the host node. Associated with GEDB is a specification of the *maximum invocation rate* (MIR) in the form of a maximum number of requests accepted for execution of the subject SvM over any interval of a specified length. Such GEDB-MIR specification of each SvM is advertised to the designers of potential client objects.

The TMO programming scheme is thus a general-style RT DC extension of the conventional object-oriented (OO) programming approach. Due to the modular nature and natural and powerful representation capabilities of the TMO model, all conceivable DC applications, whether they are RT or non-RT applications, can be designed as TMO networks.

The TMO can be supported by a middleware system compatible with widely used commercial RT OS kernels and established communication infrastructures and by the language tools compatible with established basic OO languages such as C++, C#, and Java. So far, the TMO programming has been supported without creating a new language or compiler. Instead, a middleware architecture called the **TMO Support Middleware** (TMOSM) that provides execution support mechanisms and can be easily adapted to a variety of commercial kernel+hardware platforms widely used in industry, was established [Kim99, Jen07, http://dream.eng.uci.edu/TMO/TMO.htm]. TMOSM uses well-established services of commercial OS kernels, e.g., process and thread support services, short-term scheduling services, and low-level communication protocols, in a manner transparent to the application TMO programmer. The TMOSM architecture was devised to contribute to simplifying the analysis of the execution time behavior of application TMOs running on TMOSM. Prototype implementations running on three major OS kernel platforms, Windows XP, Windows CE, and Linux v2.6, exist.
A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the *TMO Support Library* (TMOSL) [Kim99, Kim00b]. It consists of a number of C++ classes. TMOSL empowers C++ programmers with powerful and natural mechanisms for specification of unique and essential features of TMO programs. Both TMOSM and TMOSL have been used in an undergraduate course on RT DC programming at UCI for several years. Undergraduate students have found it easy to learn TMO programming (http://dream.eng.uci.edu/eecs123/learn.htm). The TMO scheme is aimed at facilitating RT DC software engineering in a form which software engineers in the vast business software field can adapt to with small efforts. The TMO programming scheme and supporting tools have been used in a broad range of basic research and application prototyping projects in a number of research organizations. New-generation application demos developed in the last few years include cars that can be driven by drivers located thousands of miles away (Figure 5), a dancing robot squad, wireless-network based digital music ensemble, high-QoS multimedia streaming synchronization, and tiled display capable of playing high-definition movies.

The TMO scheme has the potential of bringing in a significant increase in the productivity of RT DC software engineers while leading the engineers to the realization of significant improvements in the reliability of RT DC application systems produced by them as well. However, a software BB is not a fully established technological element without concomitant establishment of software engineering tools facilitating easy use of the BB. In that sense, the establishment of TMO as a 4.5G BB has still quite a distance to go.

### 3.3 Fault-tolerant DC component as a 4.5th-generation software building-block

Fault tolerant (FT) computing technology is important in realization of high-quality ubiquitous computing societies for many reasons including the following.

Figure 5. TMO networks enabling a human operator to watch the remote car through a video streaming channel and drive the remote car.
(1) Hardware component failures: While the reliability of individual computing hardware components have been continuously improving in the past few decades, the number of them employed in a sizable application system has been increasing at the same or faster rate. Therefore, in many important applications, possible occurrences of hardware component failures cannot be ignored. If a sensor or actuator owned by a node fails, then the entire node can be treated as a failed component.

(2) Flaky communication network and environment: Wireless communication channels are not quite as reliable as wired communication channels are. They are also sensitive to the environmental characteristics. As mobile computing nodes move in and out of weak communication areas, bursts of message communication failures need to be dealt with.

(3) Immature RT DC OS: Current-generation RT OS kernels and DC support middleware are immature in the area of enabling easy analysis of the timing behavior of the RT DC application software running on them. As a result, when the application software presents heavy workload close to 70 - 80% of the maximum capacity of the execution engine consisting of the networked hardware nodes, RT OS kernels, and DC support middleware, unexpected unacceptable timing behavior of the application arises at a non-negligible frequency. Unless provisions involving potentially enormous (e.g., 80%) wastes of execution engine resources are not made while using those RT OS kernels and middleware, prevention of such unacceptable timing behavior is impossible. Therefore, it is often prudent to target for better utilization of execution engine resources and make provision for tolerating such unacceptable timing behavior of some components. That is, it is cost-effective to treat such timing behavior displays as component failures and rely on RT fault tolerance techniques for overcoming the situations.

One can deduct from the above reasoning that the 4.5G RT DC component technology cannot be fully established without accompanying the 4.5G FT DC component technology. Even as our entry into ubiquitous computing societies is accelerated, the problems mentioned above will not go away. Therefore, considering the compelling needs for dealing with those problems, FT DC components, especially RT FT DC components, are important 4.5G BBs. However, establishment of RT FT DC components in sound forms is not near completion yet although some promising prototypes are expected to appear in the next several years. In fact, it is not clear if we really need FT DC component models which are good only for use in non-RT applications, once we have sound RT FT DC component models.

Sound RT FT DC components must enable easy composition of RT DC application systems which exhibit acceptable detection latency bounds and recovery time bounds. Here a recovery time bound refers to the maximum difference between a normal execution time for a task and the time for a task execution involving fault detection and recovery events.
The technical foundation which has been established so far and is relevant to the establishment of RT FT DC component models includes the following:

(T1) For detection of crashed DC nodes, *network surveillance* techniques which take decentralized, semi-decentralized, or hierarchical forms and enable analysis of tight detection latency bounds, are cost-effective [Kim00a].

(T2) For tolerating flaky communication networks and environments, mobilizing multiple types of communication networks, e.g., wireless LAN, cell phone, other radio communication networks, wired, etc., is often essential. Cost-effective techniques for coding messages which tolerate noises creeping into message bodies and the techniques for redundant transmission of messages have been well established.

(T3) For facilitating RT recovery from some component failures, replicating the RT DC components into *voting-based TMR* subassemblies or *self-checking-based primary-shadow component-pairs* is the most extensively tested effective techniques [Mos00, Kim00a, Nar02].

However various parts of the technical foundation including those mentioned above have not been seamlessly integrated yet. This situation is expected to be corrected mostly in the next 10 years.

Sizable DC application systems are built in layered forms these days. As shown in Figure 6, the OS layer sitting above the hardware layer consists typically of the kernel sub-layer and the DC support middleware sub-layer which may in turn consist of multiple sub-layers. The application software layer also consists of multiple sub-layers in many large-scale DC application systems. Therefore, it is natural to implant fault detection and recovery mechanisms in each of these (sub-) layers. Achieving a cost-effective harmoniously cooperating assembly of these mechanisms spread across multiple layers is another challenge in establishing a seamlessly integrated technical foundation. A general path to follow in such integration is discussed below.

Figure 6 depicts the types of the fault symptoms generated by each layer which can be detected at an upper layer or by the special detection mechanisms within the same layer. It was assumed that a widely used error-detection coding scheme was used in message communication and a scheme for memory-segment protection was also used. The characteristics of the faults can be summarized as follows.
Hardware layer: This layer in a DC application system, which includes both computing hardware components and communication hardware components, may exhibit crashes of some hardware components during the system operation. Transient faults of some hardware components which manifest themselves in the form of no responses or wrong value outputs (to upper layers and/or environments) for some limited periods of time, may also occur. These faults can thus be detected by the OS layer and the application software layer, which act as clients of the hardware layer, as well as by some special internal mechanisms. The faults may also lead to faulty behavior of components in upper layers.

OS (kernel + DC support middleware) layer: This layer may exhibit crashes of some kernel components and/or middleware components. In addition, transient faults in the form of unacceptably late responses or erroneous value outputs, may be shown by some components. These faults may occur due to (a) crash or transient faults of hardware components, (b) message losses, or (c) design faults that lurk within the layer, e.g., poor designs of capabilities for handling hardware component faults or overload. In turn, these faults, if not contained well in time within the component boundary, may lead to faulty behavior of components in the application software layer. They can be detected by special internal mechanisms and also by the clients, i.e., application software components. In theory, some of the faults can be detected by the underlying layer, e.g., when an OS component issues unreasonable commands which are rejected by hardware components. However, that is not a frequent situation.

Application software layer: Similar to the OS layer, this layer may also exhibit crashes of some components or transient faults in the form of deadline misses for certain actions or erroneous value
outputs. The faults may occur due to (d) crashes or transient faults of lower-layer components or (e) design faults that lurk within the layer, e.g., poor designs of action timings, etc. Whether it is worth providing special mechanisms aimed for tolerating faults of type (e), is a topic of continuing debates.

When a lower-layer component becomes faulty, it may cause multiple upper-layer components to be faulty, i.e., faults may propagate across layers. Since each fault detection mechanism may take a different amount of time for detecting a propagated fault symptom, it is possible that multiple fault detection reports may be generated within a short time-span. The approaches for handling such situations in systematic manners while maximizing the survivability of the DC application system need to be established. Here, recovery actions occurring in different layers, including retries and switches to alternate methods, must occur in harmonious coordinated manners. A sound 4.5G FT DC component must enable the application designer to design effective RT fault detection and recovery capabilities with significantly less efforts than when using earlier-generation BBs, all with quantitative guarantees in the form of recovery time bounds.

### 3.4 Secure DC component as a 4.5th-generation software building-block

Secure DC components are essentially DC components augmented with access control mechanisms and capabilities for secure encrypted communication in interacting with other DC components. Secure DC has been a highly active research field in the last 10 years. As a result, the secure DC components based on basic DC components have been much studied. On the other hand, establishment of secure DC components which are extensions of RT DC components or FT DC components is still in an early research phase without concrete demonstrations, largely due to relatively immature nature of the currently available RT DC components and FT DC components. The efforts for evaluating the characteristics of various security mechanisms with respect to their compatibility with timeliness requirements in various RT DC application environments are still under way in various research circles.

### 3.5 Mobile agent component as a 4.5th-generation software building-block

Mobile agents are software modules with the following fundamental characteristics:

1. Agents are dynamically created by their *employers* which are software modules and may themselves be other agents;
2. Agents are substantially autonomous in that they receive primarily "high-level commands" from their employers;
3. Agents may run in the sites where their employers are running but they are also dispatched by their employers to "visit" and run in remote sites whenever needs arise.
Mobile agent components are essentially DC components augmented with mechanisms such that they can be dynamically dispatched to run on the sites that are detected at run-time by the dispatcher components. The dispatcher components may be called the employer components. The programming language Java provided convenient facilities for design and dispatch of mobile agent components and these mechanisms have been incorporated into the web browser. As a result, mobile agent components have been a highly active research field in the last 10 years.

Integration of the concept of mobile agent components and that of RT DC components has been slow, however. That is, **RT (mobile) agents**, which must perform output actions in manners meeting specified timing requirements, have not been established in any mature form yet. This is again largely due to the relatively immature nature of the currently available RT DC components.

In establishing sound RT agent models, the following challenges must be met.

(RA1) **Timely admission and activation at the destination site:** Since an RT agent is subject to a deadline for its first submission of a report to its employer, its dispatch must be executed efficiently in a time-bounded manner. Admission of an agent in the remote host site involves reservation of necessary resources in the host site. Therefore, both the remote host site manager (HSM) and the communication infrastructure involved must act in time-bounded manners [Kim02d].

(RA2) **Timely interactions between agents and employers:** In general, an RT agent has deadlines to meet in making reports to its employer. The agent and the HSM must cooperate effectively so that the agent may acquire necessary execution resources in timely manners as other RT processes and RT objects in the site do. On the side of the employer, it must respond within predetermined time bounds whenever requests come from an RT agent.

Once sound RT agent component models are established, one can expect that they can be systematically extended into FT RT agent component models by mobilizing the mechanisms discussed in Section 3.3. Basically, the following requirements must be met.

(RA3) **Timely discovery of dead or sick agents and RT recovery:** Dead or sick RT agents must be discovered in timely manners by both the HSMs of the host sites and their employers located in remote sites. Examples of sick agents are the agents making unreasonable attempts to access certain prohibited entities in the host sites, the agents making invalid responses to the inquiries from the employers, etc. Once a dead or sick agent is discovered by the HSM in the site hosting the agent or the employer, the HSM and the employer must cooperate to erase any remnants of the agent in the site. The employer must then perform some RT recovery actions.
3.6 Mobile agent component as a 4.5th-generation software building-block

Web service components are DC components that interact among themselves via standardized web service transport protocols such as SOAP [Mit07] and HTTP rather than the UDP-based or TCP-based relatively thin remote method call protocols such as JAVA RMI or TMO service requests [Boo04, Kim00b, Ses05]. Their interfaces must conform to the WSDL standards [Chr01]. Since all major programming environment vendors agreed to support web service transport protocols and web sites based on a variety of platforms are capable of supporting the protocols, a great number of new-generation DC applications are expected to be built in the form of web service component networks.

While the establishment of web service component models as extensions of basic DC components has been rapid, establishment of RT web service component models has hardly begun. At present the web service transport protocols are considerably less efficient than the relatively thin remote method calls which are based on UDP or TCP and used in basic DC components are. Therefore, the RT DC applications that can be produced currently by interconnecting web service components are those tagged timing guarantees of one-second-level precision at best. Various ways for optimizing the protocol implementations to reduce overhead are expected to emerge via future research. Also, time is ripe for starting the integration of the web service interconnection approaches with the 4.5G RT DC component models discussed in Section 3.2.

4 A target 5th-generation software building-block and conclusion

A clear target for research in this area in the next decade is to establish a 5G DC software BB that possesses all the characteristics discussed in Section 2. Such a BB should enable systematic and economic construction of DC applications that are capable of taking critical actions with 100-microsecond-level or even 1-microsecond-level timing accuracy, fault tolerance, security enforcement, employing mobile agents, and operating components interacting via web service transport protocols while being easily expandable and taking advantage of all sorts of network connectivity. Only with such a BB available, a vast number of new-generation DC application software needed to establish high-quality ubiquitous computing societies can be constructed in easily analyzable and thus highly reliable forms with acceptable costs. Then realizing high-quality ubiquitous computing societies at the rate expected by the citizens will become feasible.

Core issues and promising directions in solidifying various types of 4.5G component models were presented. As progresses occur in suggested directions, their integrations will follow closely. Therefore, the first reasonable version of a 5G BB is expected to arise shortly after the last maturing 4.5G component model takes a sound form.
One of the most advanced types of RT DC component models, the TMO, was discussed in some detail in Section 3.2 to illustrate a possible starting point and a possible direction to pursue in searching for a more desirable BB. We believe that the most cost-effective way of developing other types of 4.5 BBs such as a FT DC component model and a secure DC model, is to develop them as extensions of the RT DC component model after the latter is established in a solid form. We expect that at least a few types of 5G DC software BBs will emerge in solid forms in the coming decade, including one built on the basis of the TMO.

Acknowledgment: The work reported here was supported in part by the NSF under Grant Numbers 02-04050 (NGS), 03-26606 (ITR), and 05-24050 (CNS). No part of this paper represents the views and opinions of the sponsors mentioned above.

References


