



The Social Enterprise

Unlocking the Power of Collaborative Computing

In this white paper Neeve¹ Technologies outline a new vision for distributed application design which incorporates the principles of Social Networks and Real Time Collaboration into a new computing platform.

¹ New Era of Event Driven Technologies.

Collaborative Computing Defined

With the advent of globalization, increasing data volumes and message rates the long standing challenge of *cooperative processing* has evolved into a full scale problem. Current enterprise systems remain fragmented and system resources are often hidden behind layers of complex integration software. Despite available solutions and multiple attempts at introducing interoperability standards into business process automation the task remains a difficult one. *Collaborative Computing* presents a unified data processing model in which Services, Processes and People collaboratively work on shared data towards a common goal. The collaborative paradigm offers a possible solution for reducing system complexity and solving the cooperative processing problem.

Technology Evolution

By definition, *cooperative processing* implies that two or more distinct participants complete a single transaction by sharing local computing resources such as CPU, disk or possibly a data management system. Coordination of resource use is the responsibility of the participants. The implication is that users communicate with each other and follow common rules in order to cooperatively access shared resources. This technique is related to both distributed and Client/Server computing with system participants typically executing concurrently on different processors or machines.

Cooperative processing was first introduced in the 1970's and became widely adopted by centralized mainframe computing systems in order to allow multiple business applications to function as a single *data processing pipeline*, a concept borrowed from Henry Ford. Common examples of this may be found in Airline Ticket Reservation systems, Automated Check Processing or any system wherein the same data sets are worked on by multiple participants in a specific order. Although cooperative processing has undergone a number of significant changes with the emergence of Network Computing and Client/Server in the following decades, it remains the basis for most modern computing systems.

Adoption of the cooperative approach resulted in a de-centralization of computing systems and a broad acceptance of *distributed computing* applications. In a *distributed application* interactions between users and the system appear as though all processing is taking place locally but, in fact, processing may be distributed across several remote network nodes with the user's application logic controlling the flow and sequence of resource access.

When first proposed, the distributed cooperative model presented a new and game-changing computing paradigm. However the approach relied on system participants (the users) to solve the problem of coordinating resource usage. As the paradigm gained popularity, distributed applications eventually out-paced all other architectures. Growing popularity of the Internet helped make the *distributed, cooperative* computing model a new standard in system design. Most electronic commerce sites in use today are implemented using this approach.

Unfortunately, the distributed application model has done little to address the challenges of cooperative processing. The problem of *collaborative* resource use remained unsolved since the evolving paradigm isolated participants and made coordinated resource use more difficult. As the number of shared resources and their users increased so did the scope of the problem.

Collaboration

Over the years a number of technologies have been used to address the challenge of collaborative resource sharing. One of the more prominent solutions remains the Database due to its ability to provide so-called ACID² mechanisms for transaction processing. In the database a central, structured data management system is responsible for coordinating access to shared data resources. This is achieved predominantly by locking data elements while they are in use by participants. Application Server technologies use a similar approach for coordinating access to application logic components. Resource locking may be used to enforce the so-called *mutual exclusion principle*, whereby participants competing for the same resource are put into a wait state while the resource is in use. This technique was often used by technologies such as portals and Content Management Systems as a primitive mechanism for state passing between collaborating participants.

Most recently, Message Oriented Middleware has become the preferred way for collaborative participants to communicate with each other and their resources. Messaging systems provide a communication model that is more flexible and cost effective than traditional Client/Server systems. They include state management, coordination and some data sharing capabilities offered thru the use of *sessions, transactional message passing and queuing mechanisms*. The *message passing interface* facilitates a simple, yet powerful computing model that allows shared resources and users to act as peers. This flat view of the enterprise allows participants to easily communicate with resources and each other as well as organize into peer groups.

² **ACID** (*atomicity, consistency, isolation, durability*) is a set of properties that guarantee [database transactions](#) are processed reliably. [Jim Gray](#) defined the properties of a reliable transaction system in the late 1970s and developed technologies to automatically achieve them. In 1983, Andreas Reuter and Theo Haerder coined a descriptive acronym: *ACID*.

The peer computing model has become increasingly popular with system integrators and social application developers alike. For example, the *Enterprise Service Bus* exploits peer computing techniques to enable reliable communication between services in a *Service Oriented Architecture*. This allows enterprises to link together disparate applications in an efficient and cost effective manner. Further application of the peer computing model can also be seen in popular collaboration software such as Instant Messenger, Document Sharing and Web Conferencing. Likewise, the success of social network systems like Twitter and Facebook are the result of a critical blend of capabilities offered by technologies that facilitate collaborative data processing.

At present, most popular collaborative applications focus primarily on coordinated resource sharing between individuals; allowing files, messages or media content to be passed between participants. Although such technologies are not considered general collaborative computing platforms they exploit many of the same principles and techniques found in the *collaborative computing* model. The social aspect of such systems is critical to their success as it reflects an evolution of system component interaction, as well as the changing way that businesses conduct their daily operations. The growing size of a mobile, distributed telecommuting work force also implies that the number of users, intent on collaborative data processing is growing exponentially. As technology continues to evolve, so does the scope and functional requirement of collaborative computing systems.

Notably absent from existing solutions is a product that can address the above technical challenges in an integrated and uniform fashion. Architects designing such systems today must rely on costly solutions knit together with custom code using a variety of software components such as application servers, database systems, caching technologies and messaging software. The large number of system components often requires a unique skill set necessary for integrating all supporting technologies into a cohesive development platform, reducing the likelihood of success. Furthermore, developer teams often find that they lack the tools to engage in collaborative development and implementation before an actual solution can be designed.

Collaborative Computing

The term defines an approach to cooperative data processing. Collaborative computing introduces a data-centric computing model wherein participant Services, Processes and People cooperatively work on shared data or transactions towards a common goal. The model integrates key aspects of social networking and distributed computing principles into a unified data processing platform; and identifies a number of critical disciplines that are essential to a successful implementation. It is the next logical step in evolution of cloud computing and service grid technologies providing a new foundation for designing and implementing distributed computing applications.

An important distinction is that Collaborative Computing describes the relationship between cooperative programs in a system. It is not an architectural style or product, but rather a peer computing model wherein, each participant may act simultaneously as both a client and a server, and each has equivalent responsibilities, status and global visibility.

By definition, a collaborative process implies jointly working in a group with occasionally connected participants. Consequently such group activity requires that all participants are well-defined and their availability is always known. A participant definition must therefore provide a distinct name and address to be used for all communications. This is sometimes referred to as *presence* or *Presententity* (persistent entity) and is frequently found in *real time collaboration* tools such as *instant messaging services* that have become the cornerstone of group activity.

The collaborative paradigm does not draw a distinction between real time and general collaboration since the same computing techniques may be used to facilitate both. However it should be noted that *real time collaborative computing* is already a widely adopted model currently representing millions of users. Availability of global standards such as *XMPP* has made it possible for almost any application or device to become a collaborative participant. Web applications, smart phones and even users of established technologies like Microsoft's Office Communication Server may be configured as members of the same system. When considering the benefit of integrating users, applications and devices into a unified computing system the value of a collaborative paradigm becomes self evident.

The Collaborative Computing Model

Collaborative computing draws on a number of existing principles and techniques in data processing in order to achieve its goal. The model is not prescriptive as to what technology or software components make up the system. However it does present a number of key requirements that must be met in order for a technology to be considered a collaborative computing system or platform.

Most importantly, the model requires that a *participant view* is presented which encompasses services, processes and people. All entities may dynamically discover each other, check for availability and interact by exchanging structured content using a common interface. Therefore a solution must facilitate some *reliable data transfer* between participants, an *ability to share tasks and data formats*, as well as the *ability to share data processing logic*. As such, a collaborative computing architecture spans across several data processing disciplines including Structured Data Management, Event Processing and Service Oriented technologies.

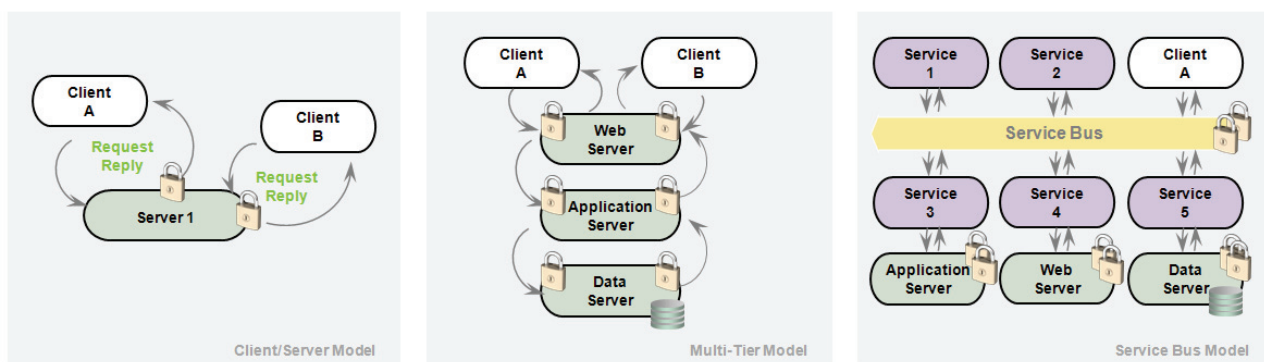
To understand the motivation behind the collaborative approach it may be useful to present several concepts that are the driving force behind the technology. The next set of topics draws on ideas that have shaped our thinking and identify several architectural issues that have led to the emerging model.

The Flat Earth Paradigm

One of the principal tenets of collaboration is the need for *awareness and discovery*. In a collaborative system participants must be able to discover each other as well as be able to understand each other's roles in the larger whole. This desire comes from decades of working with closed computing systems wherein participants and their exchange of information occurred, for the most part, in an anonymous and isolated fashion.

Early computing systems mandated a certain degree of user anonymity and isolation using this as the basis for their security model and transactional integrity. Users were not aware of each other's presence unless they occasionally failed to gain access to some shared resource such as a table row or service bean. Users encountering a locked resource could re-try the operation without any knowledge of who locked the resource or why. This approach has been taken by database systems and application servers alike.

As architectures evolved from Client/Server to multi-tiered computing and eventually into bus-based, distributed systems their function and component hierarchy grew more complex. With increasing complexity, the number of shared resources that could result in a locking situation also increased. The diagram below illustrates this by showing how the evolving paradigm increased resource contention (locking) and imposed a rigid component hierarchy on system designers.

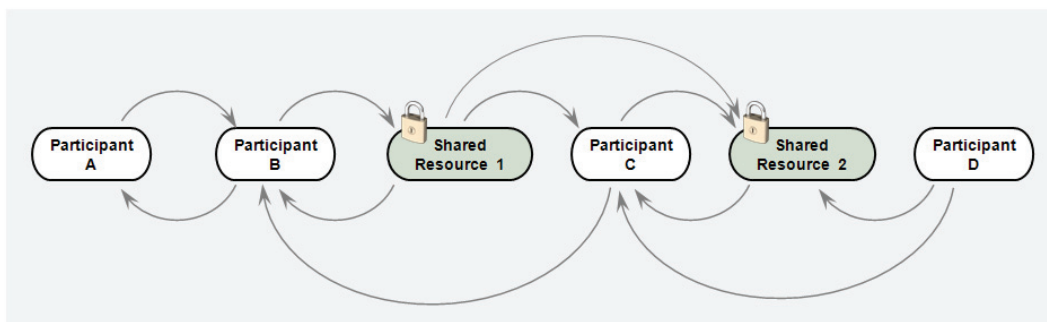


Regardless of the approach taken, resource contention continued to increase even with the introduction of mediating technologies like a service bus. Although the service bus presented an extremely useful abstraction it introduced yet another contention point for shared resource users. Furthermore all three models follow the principle of anonymous interaction making it extremely difficult to identify which participant is using the resources, although such information is critical to achieving *operational visibility*.

However, it is the component hierarchy that presents the biggest set of problems. A hierarchical system significantly lacks in resource accountability since nearly all component interactions occur in an anonymous fashion typically thru a set of pooled connections. Security and role definitions span across databases, application servers and/or messaging systems, all of which have their own security mechanisms and role management. Enforcing security or single sign-on is complex and outright impractical. Most importantly, the hierarchical model does not provide any mechanism for participants to communicate with each other or even become aware of each other's presence, making coordinated processing impossible as well. This often leads to unwarranted resource consumption and redundant implementations, driving the need for additional tools and products to fill the gap.

In many ways multi-tiered systems have become more limiting than the traditional database applications they sought to replace. Throughput and latency introduced by such systems makes them unlikely candidates for high performance solutions. We can say with some certainty that at this stage, a hierarchical approach to distributed system design has outlived its usefulness. It has become too complex and costly to maintain. Although a number of vendors offer coordination and orchestration tools to compensate for such shortcomings, the added abstraction layer frequently compounds the problems and further increases cost. A new approach is needed that will allow for a simpler interaction between participants and resources that offers performance, resource control, accountability and a more cost effective change management capability than its predecessor. This approach must incorporate global *command* and *control* capabilities into an open and collaborative computing platform.

The so-called *flat earth* paradigm proposes a peer computing model, wherein all participants are known and accountable entities. Our intent is to simplify the architecture by reducing the number of components and redundant capabilities, thereby reducing complexity, cost and risk. The approach builds on concepts found in *real-time collaboration* technologies like *instant messenger*, IP Telephony and *on-line gaming* software. It mandates that all participants be known and registered entities in order to engage in cooperative interactions. Each participant may play the role of both a request initiator (client) and request processor (server) and all participants must advertise their capabilities and acceptable data formats (interfaces). The paradigm advocates for smarter clients and a simplified communication model, thereby flattening the architecture into a set of logic components wrapped in a reliable *peer-to-peer communication fabric*. The diagram below illustrates interactions between such components.



Flat earth reduces distributed computing entities to two possible types, a *participant* and a *shared resource*. A participant is any client or logic component that may consume or produce events (messages) and exchange structured content with its peers. A shared resource is any component that, in addition to being a participant also provides concurrent access to its resources using the same communications principles. The model is not prescriptive as to the implementation of shared resources or protocols. As such it may be possible to convert data management systems or application hosting containers into shared resources. It may also be possible to use protocols such as SOAP for participant interactions. However the limited nature of Web Service technologies may not be practical for fully realizing a collaborative system's potential.

The principle motivation behind the flat earth approach is to simplify system design by making it more representative of the way people actually interact with each other and with enterprise systems. Flat earth presents a computing model that can be easily implemented by Web Browsers, Mobile Devices and Service Oriented technologies alike. Systems built on this model would become *behavior facilitators* allowing social computing principles to be applied across a variety of enterprise applications.

Thinking Locally and Acting Globally

A key benefit of implementing a collaborative system is reduction of inefficiencies in the business process. For example reducing a bottleneck in mortgage application processing or optimization of the way an asset tranche is valued. Automating a business process by implementing a Work Flow or Business Process Management (BPM) system often allows a company to reduce operational cost or make changes to an existing process faster. However, it has been observed that a large number of business process automation projects fail to deliver on their promised value because they fail to integrate one of the most critical resources of a business: the people.

Human interaction is a vital part of any business process and may take several forms. The most common interaction is a workflow that requires manual approvals, escalations or similar user participation across disparate locations and owners in order to advance the flow of process logic. Software vendors typically offer BPM capabilities as add-on components of an integration solution. This approach has a number of benefits and drawbacks. A BPM engine facilitates orchestration of process logic across a variety of components solving some of the coordination and accountability problems. However, a BPM engine requires additional integration between itself and the orchestrated components, thereby increasing the chance of implementation errors. The more interfaces and transformation steps a process has, the more error prone its implementation and change management. Hence the principal benefit of a BPM solution is in assisting the development of applications that are based primarily on human interaction, such as document work flow, self-help or web-based provisioning systems.

While BPM tools focus on cooperative data processing between users, a collaborative system must account for all manners of human interaction including collaboration between interface designers and system operators responsible for supporting the daily functions of a process. Business processes implementation often involves multiple development teams. When exceptions occur they typically result in global, cascading failures in other parts of the system. Certain process flows may halt when an error occurs and require that corrective action be taken by the user. BPM vendors do not typically provide global impact analysis or fault isolation tools, leaving this task to the system's developers and operations teams.

Experience has shown that a high percentage of inefficiency is simply attributable to human error, such as a lack of *situational awareness* or *critical thinking* that fails to consider all aspects of a given problem before implementing a solution. This is not surprising given the breadth and scope of modern business processes. With a large number of moving parts it is often difficult to take into account the global impact of a specific failure or exception and just as difficult to analyze the root cause of a problem. As such, response to failures becomes a slow and iterative process.

Social convention and human judgment also play a key role in collaborative environments. Ironically by opening up a business process to human interaction we are also increasing the likelihood that unforeseen failures occur. Consider a simple example of an XML document being created for a financial transaction. The user is working with a customer list for Latin Americas and has correctly entered in the customer locations of Bogotá – Colombia and República de Chile. The application developer correctly designed a data entry form to accept any text content. The form builds an XML document and submits it to our process. The process developer correctly designed a process to parse the XML and process the request. The system was unit tested by the individuals and deployed. Unfortunately most XML is generated with an encoding of UTF-8 and XML parsing libraries will have a problem with the special characters *á* and *ú*. Despite successful testing, once deployed, the process fails with a cryptic XML parsing error. Because the error occurs in such an unlikely place, the submitted data may actually be lost altogether.

This type of problem is not an isolated incident. With the advent of service-oriented architecture and application of so-called 'agile' development techniques, incompatibilities in data format have become the norm. The problem's cause remains the same: people. Although best practices and methodologies emphasize collaborative development, very few tools on the market actually assist in the collaborative process. While personal collaboration tools such as Instant Messenger, Video Conferencing and Document Sharing may assist in collaborative design they do not enforce data formats or assist in reliable interface definition. As a result developers often engage in function-driven design that leads to *localized thinking*, where interoperability is secondary.

Localized thinking tends to focus on resolving the immediate issues faced by a business unit and does not take into account the global impact of technical decisions. Simply put, data producers and consumers do what is convenient for them and their local system and disregard any negative effects this may have on other parts of the business process.

As a matter of practical application, the above problem could have been avoided by enforcing collaborative interface development. Developers would jointly define the format and agree upon semantics and taxonomy. The definition would be placed into a shared location intrinsic to both creation of the data entry form and process definition; for example some type of shared interface definition repository. Each participant would then be able to continue working in a localized fashion without concern for global impact of any changes they may introduce. A critical interface change would be immediately caught during *individual* unit testing.

One important goal of a collaborative system is to reduce or eliminate any global negative impact that this type of *localized thinking* may create without imposing penalties or limitations on the local system. In other words *the goal is to reduce or eliminate the risk associated with changes to a shared interface, data structure or business logic*.

Irreducible Complexity³ in Collaborative Systems

As illustrated, many distributed computing systems today make use of a multi-level component hierarchy. This architecture often leads to over-architected systems that are slow and expensive to maintain. The result is often a *loss of business agility, a lack of process governance* and a *failure to meet market demand*.

In the final analysis the root cause of the problem is a design flaw sometimes referred to as the Philosophers' Error. An architect, much like a philosopher, mistakenly assumes that the value of a system lies in the sum of its parts. Experience shows otherwise, that true value lies in the components used to build a system. The real value is in the fact that a given system can be destroyed and nevertheless, retain value as building material for creating a better system. This critical aspect of the architecture gives collaborative systems their agility.

A strong architecture, therefore concentrates on developing functional components of *irreducible complexity*. Such components, frequently referred to as services, are independent units of logic designed to perform a specific task and behave in a uniform way

³ Irreducible complexity in software design as applied to System Theory. Our use of the term does not in any way reflect religious connotations of the concept. However, we will use the same definitions of an irreducibly complex system as one "composed of several well-matched, interacting parts that contribute to the basic function, wherein the removal of any of the parts causes the system to effectively cease functioning".

across implementations. System components are said to be irreducibly complex if the act of altering any functionality or aspect of a component renders the system dysfunctional. Consider a typical participant in a collaborative system. Every participant component must know how to communicate with others, process requests and register itself in a participant roster or group. Eliminating any of the stated functions breaks the collaborative nature of a system.

Developing components that are irreducibly complex can be somewhat challenging. In order to be effective an implementer must overcome the instinct of *functional encapsulation* that is the hallmark of proper object oriented design. In other words developers that design services frequently make the mistake of overloading their components with unnecessary functionality that is often duplicated in other components.

Consider a service that executes SQL queries in a database. Such a component would have to be able to (i) receive a request for query execution, (ii) obtain the query to execute and (iii) be able to supply login credentials and session information to the database. We can encapsulate the first two functions into a message or event that holds the SQL query. In fact a number of implementations of such a service on the market today do exactly that. However a more careful problem analysis shows that encapsulation will force the process designer to build a new SQL query object every time the service is invoked. This adds an extra step to each SQL query call and limits component reuse in cases where static SQL is used. As such, the proposed encapsulation does not make sense. On the other hand, session settings for each database tend to be different and should probably be separated from login credentials.

One of the main objectives of collaborative systems is to produce the simplest possible design without sacrificing functionality. The least common denominator in the system is the participant or client. In the collaborative model a participant component would be implemented using principles of irreducible complexity. A participant's function library would contain all the critical capability required for collaborative communication, discovery, state coordination and role advertising. While this makes for a 'thicker client' and a more complex design, the component model is greatly simplified leading to a more efficient, agile and cost effective solution.

The Social Enterprise

The collaborative model combines multiple data processing techniques into a broad and flexible computing platform. This approach can facilitate reliable, cost effective solutions for system integration, collaborative applications, data distribution and management across an enterprise.

By their nature, collaborative systems are social networks built around users, logic services, mobile devices and business processes. Members can dynamically form collaborative communities without regard for geographic location, application language or individual data format. Most notably, the goal of collaborative computing systems is not to replace traditional business applications. The intent is to borrow concepts from social computing technologies in order to build more agile and effective enterprise systems.

It has been observed that any technology used to facilitate electronic commerce strives to emulate an interactive market wherein bartering and exchange of goods and services can occur in a reliable fashion. If our business process is the *data factory*, then our *product* is data and our *added value* is information about the data's current state. Collaborative applications take advantage of social computing principles inherent to all market systems, thereby allowing the technology itself to act as an electronic market place for data. Instead of building such systems from scratch, developers now have the option of buying them.

Once established, collaborative systems often promote the evolution of new collaborative exchanges for goods, services and ideas because all markets tend to grow organically⁴. The goal of a collaborative computing platform is to provide a way for information producers and consumers to come together in a secure and potentially anonymous way without sacrificing governance or visibility into such systems. This is true of social networks as well as corporate enterprise systems. Consider the *intellectual assets* of any IT organization. Experience shows that as much as 50% of all critical business information may be locked in personal collaboration tools like smart phones, E-Mail and Instant Messenger. While social convention promotes the use of such tools the critical content being shared between users is frequently lost or inaccessible to the business.

Collaborative systems can facilitate a seamless integration between personal collaboration tools and enterprise systems allowing users to tap into a broad range of enterprise capabilities and data without altering their social habits. Participants may globally advertise their function, availability and intent as well as be able to discover other members or resources and the types of data they produce or consume. The result is a *social enterprise* built around the intellectual assets of employees, customers and other contributing members.

Invoking the frequently cited analogy of the Gold Rush of 1849, the folks who turned a real profit during the Gold Rush were the ones selling prospecting tools such as picks and shovels, not the ones who went digging for gold. Embracing collaborative computing as a new model for application design presents an opportunity for technology manufacturers to supply the next wave of prospecting entrepreneurs with tools to find their gold.

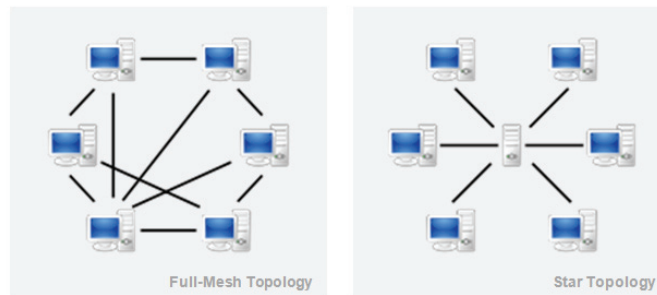
⁴ Consider the emergence of Twitter and the cross-linking of blogs and dating web sites to Twitter and Facebook as examples of evolving collaboration exchanges.

Key Principles of Collaborative Computing

The collaborative model identifies a set of principal tenets⁵ that define the scope and function of a system. The model is not prescriptive as to the specifics of implementation or how the functionality is achieved. However a given system must satisfy all of the following technological requirements in order to be compliant with the definition of collaborative computing.

Peer-to-Peer Interaction

All participants in a collaborative system are considered peers. Peer computing is defined as a distributed application architecture wherein all members are equally privileged, *equipotent* participants. They are said to form a peer-to-peer network of nodes. Peers make a portion of their resources, such as processing power, data or network bandwidth, directly available to other system participants, without the need for central coordination. Peers are both producers and consumers of resources, in contrast to the traditional Client/Server model where only servers produce, and clients consume. Peer systems typically implement one of the following topologies depending on the goal and purpose of the application:



Full-mesh systems, sometimes called *single-hop routing* topologies do not have a centralized infrastructure. Such topologies are often found in file sharing and telephony networks. Participants are always equipotent and the network system is said to be *self-organizing* as it does not have a central governing mechanism.

The star topology utilizes a central coordinator model which may provide registration or routing services for participant nodes. Members typically depend on the coordinator for certain capabilities and may not be *equipotent*. Popular implementations include Web Conferencing and Instant Messaging systems. The approach is also used by message brokering software, sometimes referred to as store-and-forward message systems.

Collaborative systems may be designed using either approach depending on the specific implementation of routing, data distribution and discovery techniques. Some of the more prominent technologies such as WebEx and Kazaa may implement a hybrid topology as this allows for a more flexible and feature-rich solution. Peer-to-peer (P2P) systems often implement an abstract overlay network, built at the application layer on top of a physical network. Overlays are used for routing, discovery and traffic isolation making the system independent from its physical network topology.

👉 Peer interaction (*equipotence*) is mandatory for collaborative system design as it provides a foundation for all other functionality.

Membership and Participant View

A view of members or participants provides facilities for seeing and discovering components that are part of a collaborative system. Information is typically presented as a lists or roster of members in the system or a particular set of components (group). For example, personal collaboration tools such as Skype allow users to select from a global view of all members and add them to a private list. Users may create a group of conversation participants from a private list. The purpose of a *membership view* is to provide information about the state of participants; for instance, *ready*, *available* or *busy* are popular participant states. Views are dynamic and notify the observer when membership changes occur such as when members join, leave or transition to a busy state.

The collaborative model presents a *generalized participant view* that encompasses services, processes and people. All entities may dynamically discover each other, check for availability and interact. Users may check on the status of service components or business processes. Participants may be organized into groups and their state changes are presented as discrete events. Participant *presence* or a *change in state* may in turn drive business process logic. For instance, a set of services that form a process may notify observers when the state of various components changes advising on the general health of the process.

⁵ Our model would not be complete without a nifty set of acronyms that allow users to identify the principal tenets of Collaborative Computing Systems. The key properties of a collaborative system are Equipotence, Consistency, Isolation, Accountability, Pertinence and Persistence (ECI/APP), pronounced ez-app.

The concept of entity presence introduces a fundamental change to the old computing paradigm. Unlike conventional systems wherein participant nodes implement extra routing layers to obscure the identity of participants, collaborative computing promotes the use of *distinct addressing*, *global visibility* and *direct communications*. Rather than engage in anonymous data exchange all components must make themselves and their state known, leading to a more transparent and reliable system.

👉 In a collaborative system participant views are essential for *accountability*, resource governance and state reporting.

Group (Targeted) Communications

Targeted communication is a critical aspect of collaborative computing systems as it allows network interactions to be localized to a particular set of participant components. This type of network *traffic isolation* allows participants to form virtual work groups and engage in localized, cooperative data processing.

Group communication plays a critical role in collaborative system design. Unlike traditional Publish/Subscribe mechanisms, targeted communications limit the scope of *network data* visibility, allowing participants to engage in a secure and private data exchange. Grouping works together with participant views, potentially allowing members to engage in conditional behavior. For example, producers of a given event or message type may only produce specific content if there are active consumers for such data.

Network protocols that support group exchange must also provide facilities for strict message ordering allowing the system to engage in proper distribution of data. Group communications must guarantee that events or messages within a group are received by all group members in the same order they were produced; otherwise participants cannot have a consistent view of their system.

👉 Collaborative systems make use of targeted, group communication in order to achieve data *consistency* and *traffic isolation*.

Content-based Addressing

In a Publish/Subscribe system participants exchange messages using a technique called *subject-based addressing*. Producers publish data on a logical channel such as a *subject* or *topic*. Each message must have a subject name which is considered the destination address. The *subject* or *topic* name is then used by consumers to subscribe to content at a specific address. Message content is opaque. There is no correlation between a subject and its payload or structure.

Managing subject lists and the relationship between subjects and content is the responsibility of the programmer. Messaging systems do not provide a way to infer payload structure or contextual semantics of a message by examining its subject or topic. Since there are no restrictions on content produced by a publisher, subscribers must always verify the structural integrity of the content they receive. Messaging presents an honor-based system of anonymous data exchange and does not guarantee data compatibility between participants. The resulting applications are inherently complex and error-prone. Much of the logic consists of packaging structured data for transmission, un-packaging such data upon receipt and verifying its structure. As the number of participants increases the system becomes difficult to manage and debug.

Collaborative computing advocates for a *content-aware* and *moderated* exchange of structured data objects. All data exchanged between participants must have a known and well-defined structure. In contrast with messaging systems, data objects exchanged by participants should have a discrete channel name that uniquely identifies the content. The system should maintain a list of all relevant channel names, allowing consumers to look up an address where particular content may be obtained. Producers on a channel are restricted as to the scope and structure of the content they may publish. Structural compatibility of data exchanged by participants is guaranteed. This approach is sometimes referred to as *content-based addressing*.

A moderated exchange introduces the notion of a separate control system that engages in *passive governance*. A moderator does not inhibit or control exchange of data between participants. Its main function is to collect and present information about participants allowing members to discover each other by the content they produce or consume facilitating further accountability.

Data exchange between participants in a collaborative system is said to be *content-aware* as content may be used to route and filter information flow. For this reason collaborative systems prefer to treat data units as *events* rather than messages. Events are distinguished from messages by several characteristics. Event content tends to be structured, whereas message content is opaque. Certain events are also meant to propagate across nodes, essentially passing thru participants and preserving the chain of causality. Such events are said to be idempotent. They must retain their identity (ie. unique identifier or time stamp) and be capable of re-transmission by participants. In contrast, messaging systems allow users to create a new message from old content but do not allow re-transmission of a received message.

👉 *Content-based addressing* allows participants to discover each other based on the type of data content they produce or consume. It facilitates a *content-aware* exchange of information allowing collaborative systems to act like data markets, because knowledge of channel content allows data producers and consumers to identify each other's *intent* and make informed decisions.

Shared Data Management Facilities

The Collaborative model is not complete without facilitating cooperative processing of structured (or semi-structured) data. Given the distributed nature of collaborative systems it is expected that an implementation provide data storage capabilities that allow participants to engage in collaborative data management across a distributed system.

Although there are many ways to provide structured data storage in a collaborative environment, a number of facts warrant consideration. In recent years software vendors have attempted to solve the problem of *distributed data management* using a broad spectrum of technologies. However, the goal of achieving *distributed, scalable and low-latency* data systems has remained elusive. Lack of success is driven by several key factors: (i) a failure to integrate structured data management with event-driven computing concepts, (ii) favoritism towards relational data and Structured Query Language (SQL). This is not surprising given the fact that *Structured Data Management* has been dominated by database vendors for more than 20 years.

Although SQL provides great value in defining and discovering runtime data relationships, indiscriminate application of relational theory can have a crippling affect on performance and scalability. Set processing languages⁶ typically work on tuples or tables restricting their use to relational or pseudo-relational data structures. Ad hoc queries and data definition often require in-depth knowledge of SQL making such languages impractical for business users and developers unfamiliar with SQL or relational theory.

Collaborative computing favors *domain specific languages* as an abstraction mechanism from underlying data structure. Although it is expected that data storage is available as a shared resource to system participants, such data need not be relational. Within a collaborative system the community should socialize taxonomy, define data semantics and drive data distribution. This allows for optimizations such as data co-location, relationship and state keeping structures to develop naturally based on the system's usage, rather than being forced by a pre-defined data schema. As such, *alternative data management systems* (ADMS) are preferred.

👉 Data storage (persistence) is a critical resource that facilitates data sharing and state passing. There are strong arguments for and against the use of relational technologies and SQL as mechanisms for storage and retrieval of data in a collaborative system.

⁶ Set processing includes all SQL dialects, Query by Example (QBE) or any similar language that engages in manipulation of structured data arrays.

Conclusion

In this paper, we have presented a new data processing model called *Collaborative Computing* wherein Services, Processes and People collaboratively work on shared data towards a common goal. The approach offers a possible solution for reducing system complexity and addressing the challenges of cooperative data processing, potentially providing a new foundation for designing and implementing distributed computing applications.

Collaborative Computing systems are identified by several key properties:

- *Equipotence* of participant components, implying peer-to-peer interaction
- *Consistency* of data exchange between participants, ensuring proper ordering and data delivery
- *Isolation* of communication traffic and scope between participant groups
- *Accountability* of participating system members, facilitating membership views and roles
- *Pertinence* of structure and content (content awareness) relative to data exchange between participants
- *Persistence* of data-in-transit being worked on by participant components

Consider the impact and role of currently available personal collaboration tools. People may search, find each other, communicate using shared props and view information in a common environment. Participants and the information they use form a physically shared (virtually co-located) "data space".

Users can interact with each other directly or by manipulating data space information in a coordinated, globally aware fashion. All activities are persisted and synchronized enabling a common, semantically coherent environment in which all participants are situation-aware and can interact in a moderated and productive fashion.

Collaborative systems allow the same social computing principles to be applied across a broad variety of enterprise applications, transforming knowledge of personal computing into a transferrable skill set that may be applied to any business or industry.

Key Principles

Equipotent Participants

Communication Consistency

Network Traffic Isolation

Participant Accountability

Content Pertinence

Data Persistence

This same interaction model may also be applied to software components facilitating a unified, *pervasive computing environment* that encompasses users, applications, enterprise systems and mobile devices. In addition to lowering cost and improving interoperability, this approach reduces the learning curve for users and lowers the barrier to entry for system developers.

The term *Collaborative* defines relationships between cooperative programs in a given system. Collaboration is based on a peer computing model wherein participants may act simultaneously as both a client and a server, with each member having equivalent responsibilities, status and global visibility.

The paradigm builds on existing technology concepts in order to achieve its goal. It is not prescriptive as to the implementation specifics. However, a given architecture must

follow several key principles as defined above, in order to be compliant with the definition of a collaborative computing system.