Distributed Collaboration

Prasun Dewan[1]

1. Introduction

This chapter scopes out the subject of our study. It defines and motivates the field of distributed collaboration. It shows how this field relates to virtual reality, distributed, real-time and mobile computing, databases, security, and various subfields of artificial intelligence – natural language understanding, agents, machine learning/data-mining, and computer vision. It gives an overview of the area from four different points of view: applications of distributed collaboration, the issues raised by it, how it builds on related disciplines, and important research systems and products in this field.

Computer Supported Collaboration

The term distributed collaboration is very general, and includes, for example, communication via postal mail. In this course, we will focus on computer-supported distributed collaboration. Collaboration without computers can influence computer-supported collaboration by providing, for instance, real-world metaphors for computer concepts. However, we will mainly focus on our intuition/experience, rather than formal studies, to understand how collaboration without computers is carried out.

We will look at both collaborative applications and infrastructures for implementing these applications.

Collaborative Applications and Coupling

It is possible to define a collaborative application too narrowly or broadly. For instance, one can define it to be an editor that allows multiple users to compose a document together. However, this definition rules out, for example, an application that allows distributed presentations. We can be more general and define it to be an application that allows multiple users to work towards a common goal. However, this definition is too abstract in that it is not sufficient to objectively distinguish collaborative applications from non-collaborative applications. Here is a more precise, system-based definition that is consistent with the definitions above: A collaborative application is a software application that (a) interacts with multiple users, and (b) couples these users, that is, allows each user's output to be influenced by some input of another user. Input is any event (e.g. keyboard, pointer, gesture, speech) generated by a user that is interpreted by the application, and output is any event (e.g. screen update, audio output) generated by the application that is perceptible to a user.

Figure 1 illustrates the notion of a collaborative application. This application interacts with three distributed users – one of them is using a desktop while the others are using two different mobile computers. When user, U1, asks for a red circle to be drawn, the circle is drawn on all computers. Thus, this application meets the requirements of interaction with multiple users and coupling.

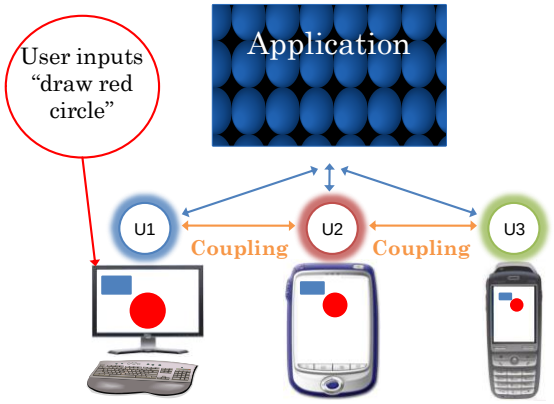

Figure 1 Collaborative Application

It is possible for an application to interact with multiple users but not couple them. A time sharing system is an example of such an application – the goal of this system is to give multiple users of a computer the illusion that they have sole control of the computer. A collaborative application, on the other hand, makes users explicitly aware of their collaborators through coupling.

Towards and Beyond "Being There"

Collaborative applications enable Computer Supported Cooperative Work (CSCW). Why provide computer support for collaboration when people have been working together for ages without the computer?

Towards Being There: Computer tools can simulate face-to-face interaction among distributed users, giving them the illusion, to different degrees, of "being there" in one location. GestureCam (Figure 2) provides a novel approach to providing this illusion. For each remote collaborator (currently the systems supports only one) it places a surrogate device at the local site that can be controlled by the remote user. The device has three degrees of freedom, carries a camera and a laser pointer, which points in the same direction as the camera. The remote collaborator can move the camera in three dimensions to view and point at different objects in the room. At a conference presentation of this device, the remote user made his surrogate device do a Japanese bow to the local user!

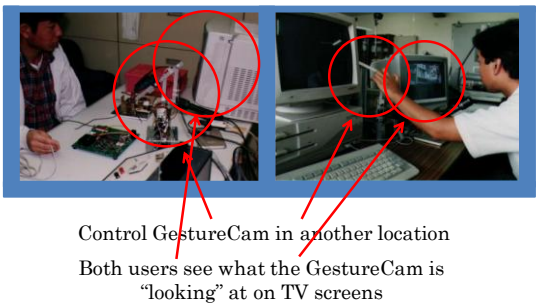

Figure 2 Towards being there with remote surrogate

While it is possible to approach being there, it is not possible to completely provide the feeling of actually being there. If all collaborative applications did was simulate "being there," then they would always support meetings that are inferior to the real face-to-face meetings. They would be the second-choice, taken when people do not have the money or time to travel to be at a common physical location. For this to happen, if they must go "beyond being there," offering features not available in collaboration supported without the computer. Users of virtual environments often want to teleport to a location rather than virtually walk to it. We need analogous features in collaborative applications.

The video browser of Figure 3 shows how this can be done. It allows users to asynchronously view the video of some live event. Naturally, not being there at the event dilutes the experience in many ways. However, the browser provides several compensating features not available to people who actually attended the event. Like a regular video player, the browser allows users to pause, rewind and fast-forward. It provides several additional, beyond being there features including video bookmarks, pause removal, time compression without changing pitch, automatic generation of slide summaries and table of contents.

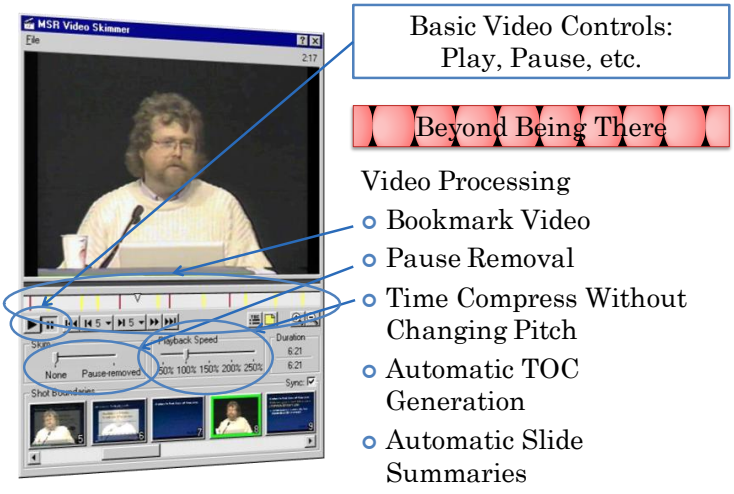

Figure 3 Going beyond being there with meeting browser

In summary, the two goals of collaborative application is to give users the illusion of being there in a common location and go beyond being there. The remote surrogate and video browser examples

focused on one of these goals. Several applications try to simultaneously meet both of these goals. The 2D chat application of Figure 4 is an example of such an application.

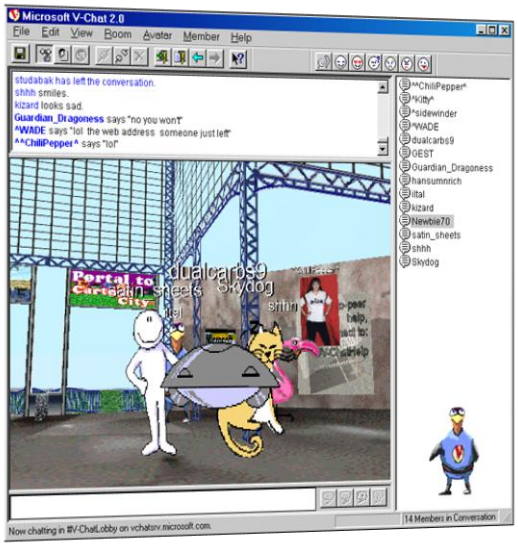

Figure 4 Combining both goals through virtual chat room, avatar selection and anonymity

The “towards being there” goal is provided by avatars, which transport users to a common virtual world. The “beyond being there” goal is met by allowing avatar selection and anonymity.

We will later see a variety of ways to support and mix these goals.

Relationship with other fields

Virtual Environments

Transporting users to a world other than their physical environment is, of course, the goal of virtual environments. A special case of such environments are immersive virtual environments, which provide head mounted displays and tracking systems to allow users to translate physical actions such to analogous actions in the virtual world. For instance, one can walk through the model of a ship before it is actually built. Remote virtual environments – environments that simulate an actual physical location – is another subclass of virtual environments that is of particular interest in the area of distributed collaboration.

Figure 5 shows the relationship between collaboration technology and virtual environments. Email and other technology for asynchronous interaction such as the meeting browser of Figure 3 do not address the “being there” goal, and thus do not create a virtual environment. The chat application of Figure 4 does create such an environment – however, the environment is not an immersive environment, nor is it a remote environment. An environment that allows multiple users to walk through a ship model would be an example of an immersive collaborative application. A video conferencing application is an example of a remote collaborative virtual environment. However, such an environment is not immersive. To provide a remote immersive environment, we would need a sea of cameras at a meeting

location, and use the images captured by these cameras to create a virtual environment in which remote users would navigate. Assuming such environments can be created faithfully in real-time, a remote specialist doctor could use such an environment to examine a patient from all angles assist some local general practitioner in surgery. This is a yet to be realized dream of some, in particular Henry Fuchs of UNC!

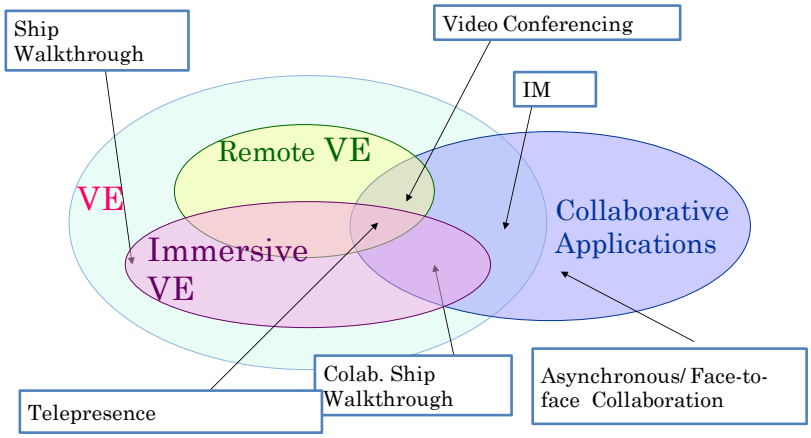

Figure 5 Relationship with Virtual Environments

Distributed and Real-Time Applications

The traditional driving problems for distributed computing have been non-collaborative. Examples include distributed file systems, concurrent execution of parallel algorithms such as finding prime factors, and more recently, and related to distributed collaboration, remote desktops.

The vast majority of collaborative applications are distributed – consistent with the goal of being there. However, there are some exceptions. Some applications allow users to use different devices connected to a single computer. Some of the popular computer games are examples of such applications. Researchers at Xerox have built at multi-mouse whiteboard that allow, for example, two users to stretch an object in two different directions simultaneously and play tug of war as they move an object in two different directions. There has been great interest recently in non-distributed tabletop collaborative applications built on top use of special multi-user touch sensitive devices that can identify which user is touching where.

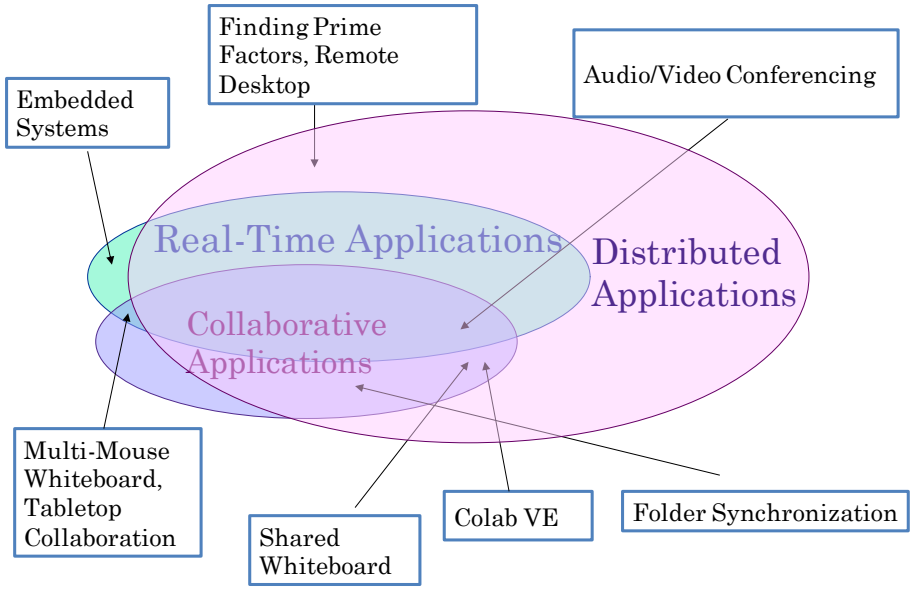

Figure 6 Relationship with distributed and real-time computing

Tabletop interaction is also an example of a synchronous application wherein users are collaborating in "real-time." It is, of course, possible to have real-time applications that are not distributed, such as an embedded system supporting anti-lock brakes; or asynchronous collaborative applications, such as an email system.

Mobile Computing

It is difficult to support real-time collaboration when a mobile computer is disconnected. In single-user interaction, disconnection leads to the need to synchronize the personal information on mobile devices with other computers replicating the information. In collaborative interaction, it is also important to synchronize information with collaborators sharing the information. Mobile devices are increasingly connected to the internet continuously or for long periods of time. In such a scenario, they can participate in real-time collaboration. Given that many more have mobile computers than traditional desktops, for collaboration to really make an impact, collaboration cannot become pervasive unless it involves mobile devices. This goal requires us to address several issues specific to mobile computers – small size, relative slow CPUs, and power conservation. Can we really imagine giving a distributed power point presentation to or from a cell phone? Some new advances address some of these issues. For example, it is possible to attach full size keyboards to mobile devices. Even more interesting, portable projectors are now available to project their displays on large surface areas. Thus, mobile collaboration seems to have great unrealized potential.

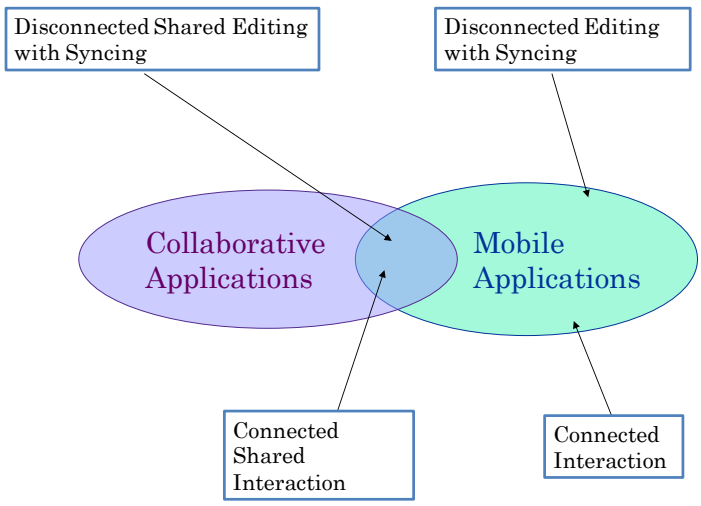

Figure 7 Relationship with Mobile Applications

Databases

While it is possible to create databases for personal use, it is more common to share them with others. Applications that access such databases meet our definition of collaboration applications, as updates made by a user can be seen by others. However, not all collaborative applications are database applications. The reason is that databases allow users to *explicitly* commit a sequence of operations atomically to persistent storage in a transaction. Many collaborative applications such as IM do not support any form of transactions. Others weaken the properties of transactions by, for instance, allowing *implicit* commitment of users' transactions if they have been inactive for some period of time. While this makes sense in several collaborations such as document editing, it does not for traditional databases such as bank databases. How transactions should be weakened in various forms of distributed collaboration remains an active research area.

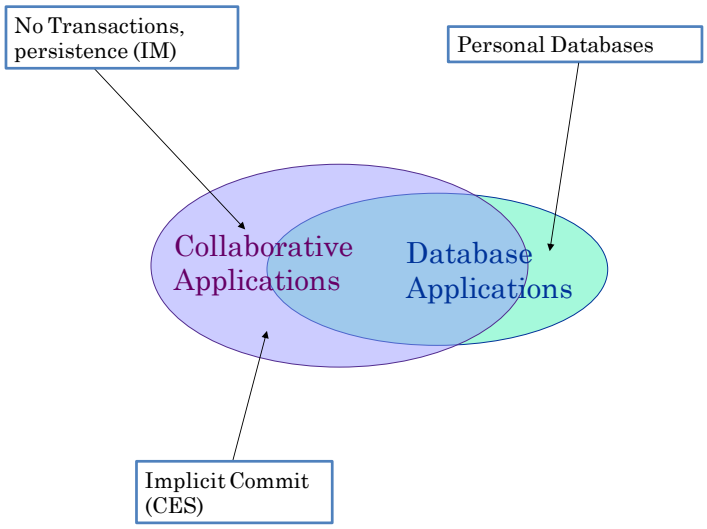

Figure 8 Collaborative vs. Database Applications

Protection/Security

Security is the popular term today for protection of our computers from undesirable actions. It can be argued that without collaboration, there would be little need for such mechanisms. If we were to isolate our computers, disconnecting them from the internet, we would not have to worry about malicious actions of others. We would still need mechanisms to protect ourselves from accidentally destroying information. For instance, we would need to make a finalized document read-only, so that we do not accidently modify it while viewing it. We need to connect to the internet for two reasons: 1) to access distributed private resources such as a remote desktop, and (2) to share information with others. In both cases we need protection from others. As (2) is more common than (1), and one needs protection from others more frequently than from self accidents, one can argue that collaboration is the main motivation for the field of security. Yet there are many collaborative applications that do without some important forms of security, as protection mechanisms can reduce the level of participation. An important example is the Wikipedia, though its designers are now considering access control to prevent slanderous comments. One of the biggest challenges in this area is to have sophisticated security while supporting lightweight, fluid, and highly-inclusive collaboration.

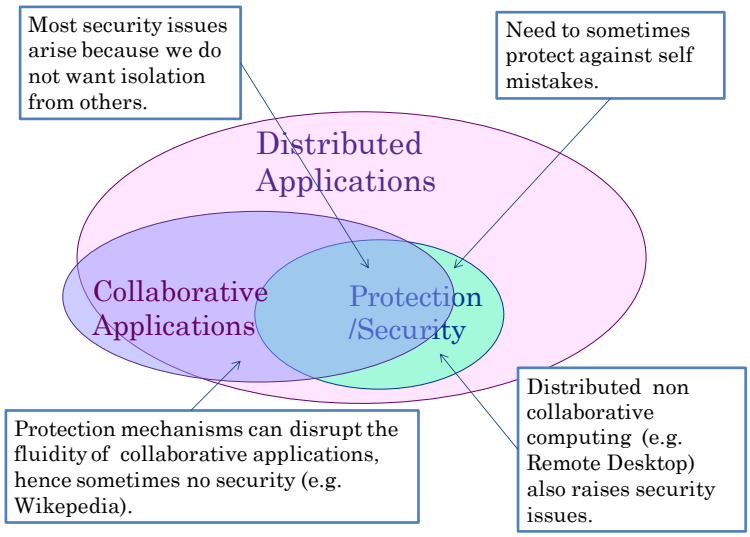

Figure 9 Relationship with Security

Artificial Intelligence

This is a broad are including several fields adding “intelligence” to computers.

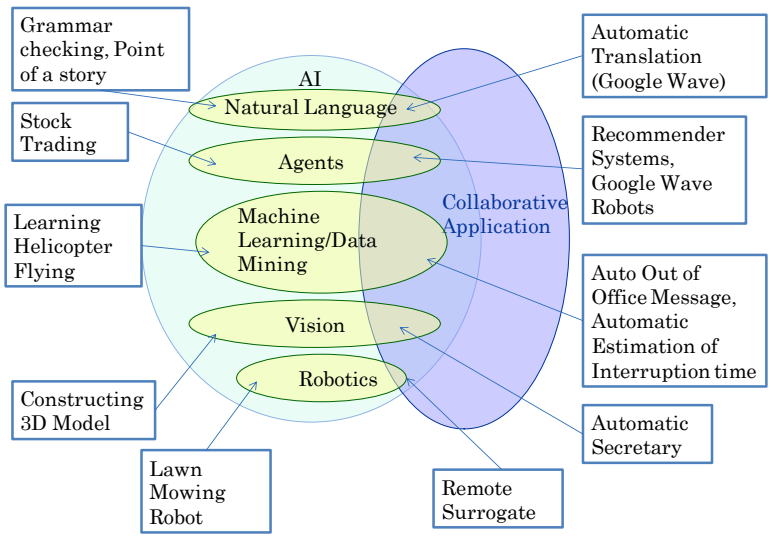

Figure 10 Relationship with various AI subfields

There has been work underway for a long time to make computers understand natural languages. We see the fruits of such work in the spelling and grammar checkers/correctors available today; though some early goals such as automatically understanding the point of a story remain elusive. There is a large group of researchers developing software agents that make decisions, such as stock trading, that we typically associate with humans. Machine learning/data mining allows computers to dynamically learn knowledge that was not programmed into them such as the art of flying a helicopter. The field of vision focuses on learning from 2-D images. An important class of such work is construction of a 3-D model of a physical object from images of it. Robotics addresses construction and movement of robots that can navigate in the physical world and perform physical actions we associate with humans such as mowing a lawn.

All of these fields have had some linkage with collaboration. Google Wave includes a tool to communicate with each other using different languages, automatically translating between the two languages. Several systems have agents that make recommendations to a user such as the book they should buy based on actions of others. Google Wave has a framework that allows software agents, called robots, to participate in a collaborative session. The remote surrogate device mentioned earlier is an example of a physical robot that fosters collaboration. Recent research has used machine-learning to predict when users are interruptible. Similarly, researchers at Microsoft have used machine learning to automatically create out of office messages that estimate when the users would return to their office. They have also used computer vision techniques to create an automatic secretary that recognizes office visitors and gives them visitor-specific information.

Integrating work in various AI fields with collaboration is a perhaps the most unexplored and promising research direction today.

User-Interface and Object-Oriented Technology

On the other hand, the most explored cross-area relationship involving collaboration is that between it and user-interface technology. This is not surprising as collaborative applications are interactive – the

main difference between them and traditional interactive applications is the number of users interacting with the application. User-interface research has yielded window systems, toolkits, and frameworks, which have been extended to support multiple users.

A strong co-relationship exists between user-interface and object-oriented technologies and the former is often the driving problem for concepts invented in the latter. An example is the model-view-controller framework we will study later, which has been extended to support multiple users. We also see that the concept of a Bean object – an object offering well-defined getter and setter methods to retrieve and change state – has been used to generate single-user interfaces in the user-interface domain and multi-user interfaces in the collaborative domain. The field of user-interfaces addresses not only the implementation of user-interfaces but also its design. There has been work in extending principles addressing the design of interfaces involving a single user to those involving multiple users.

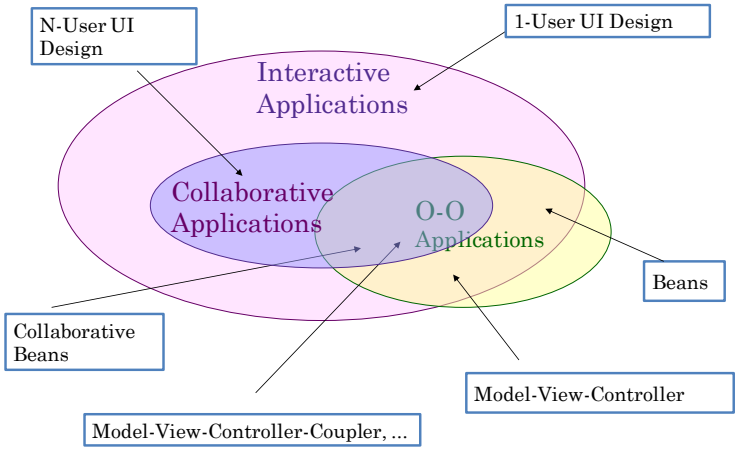

Figure 11 Relationship with User-Interface and Object-Oriented Technology

What, How, and Automation

UI Design

The user-interface of the SubEthaEdit programming environment shows that the user-interfaces of collaborative applications are, in general, different from those of single-user applications.

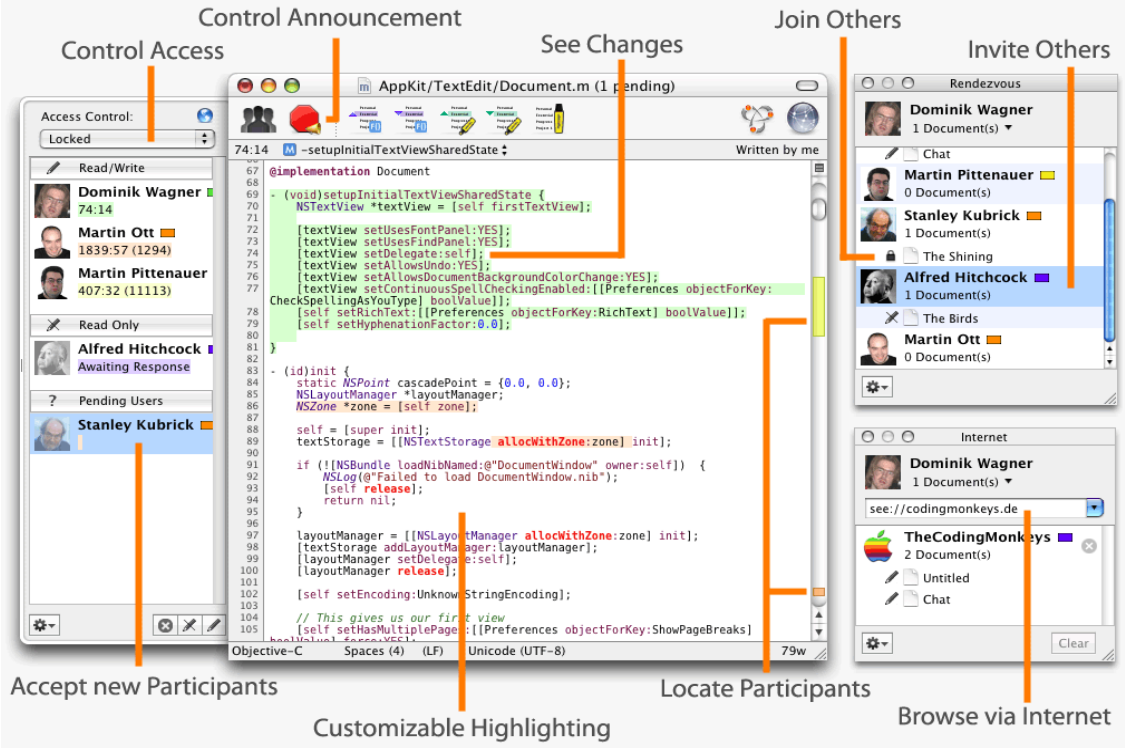

Figure 12 SubEthaEdit Collaborative Programming Environment

SubEthaEdit is a collaborative programming environment. We will later study some of the concepts behind the user-interface shown above. The point to make now is that a multi-user interface can be substantially different from a single-user interface – all windows other than the large middle window make sense only in the multi-user case and even the middle window has aspects such as highlighting and scrollbars that are specific to collaboration. The challenge in the design of collaborative interfaces is two-fold: what special functions should be provided to support multi-user interaction and how should we allocate scarce screen space between supporting these and traditional single-user functions.

Implementation

Donald Knuth wrote a famous book with the title “Algorithms + Data Structures = Program” arguing that the two main conceptual difficulties in software implementation are in appropriate data structures and algorithms. Since he wrote the book, more than thirty years ago, software architecture has become another pressing concern as the size of programs has exploded. All three issues are of great concern in the implementation of collaborative applications, because of the complexity involved in handling distributed, concurrent interaction. The figure shows that among other concepts, we will study algorithms and data structures to consistently handle concurrent streams of regular and undo commands, and architectures that create distributed versions of the model-view-controller framework mentioned earlier.

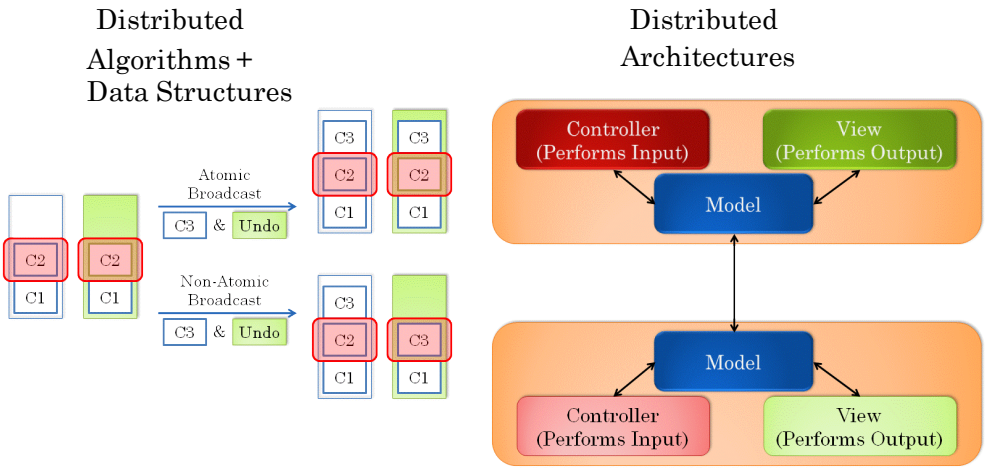

Figure 13 Collaborative Programs = Distributed Algorithms + Data Structures + Architectures

Infrastructure

Certain data structures, algorithms, and architectures apply to a large class of collaborative applications. These can be automated in a common software infrastructure shared by these applications.

The reason why applications keep getting more complex is not because we have become more productive but because increasingly application-independent concepts are being implemented in infrastructures such as programming languages, operating systems, and user-interface toolkits. These infrastructures provide us with software abstractions, layered on top of the hardware, that are of use to a variety of applications. Some commonly-used abstractions include arrays, records, files, concurrent threads, textboxes, and sliders. The underlying infrastructures implement the data structures, algorithms, and architectures needed to support these abstractions; and the applications simply use these abstractions.

A collaboration-specific infrastructure can provide a variety of automations. It can automate coupling and associated collaboration functions we will see later. The success of a synchronous collaborative application depends perhaps less on the functions it provides and more on its performance. If any of its users finds the response time bad, the whole group might have to abandon its use and resort to some other form of collaboration. Therefore, an infrastructure often automates techniques for offering acceptable performance. Those of you who have used Eclipse plug-ins know that the usefulness of allowing extensions to be added easily to a complex system. Moreover, just as we have today have multiple programming languages, toolkits and operating systems, we also have multiple collaboration infrastructures. Thus, a collaboration infrastructure can include mechanisms to easily add extensions to it and interoperate with other collaboration infrastructures.

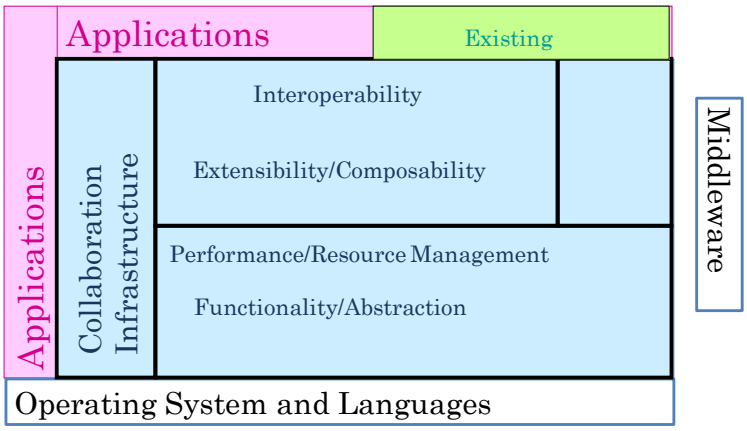

Figure 14 Components of Collaboration Middleware

A layer that sits in the middle of applications and traditional operating systems and programming languages is called middleware. A collaboration infrastructure is an example of middleware. To the operating system and language, it is another application that makes use of the abstractions provided by them. To the collaborative application, it is an infrastructure, providing its own abstractions. Some collaboration infrastructures have been integrated with programming languages and operating systems. Even these can be considered as layers on top of traditional programming languages. More important, as we see later, the integrated systems have both practical and theoretical problems, which have made them unattractive so far.

A collaborative application may bypass a collaboration infrastructure, directly using the facilities provided by operating systems, programming languages, and other relevant forms of middleware such as toolkits. Applications that make use of a collaborative application can be classified into collaboration-unaware and collaboration-aware applications, based on whether they are aware that multiple users interact with them. It is particularly important to support collaboration-unaware applications, at least in the short run, as it allows us to share existing applications.

The main challenge in the design of any infrastructure is how to offer automation without reducing flexibility, that is, without constraining the design of applications. This is particularly a problem in the case of collaboration infrastructures for three reasons. First, the design space of collaborative applications is still a matter of research. Second, these applications are interactive, and as, for instance the evolution of Microsoft Windows shows, our expectations from user-interfaces keep evolving. Third, in the case of collaboration-unaware applications, an infrastructure cannot delegate design decisions to the applications. This is not the case, for example, in the case of operating systems, which focus on providing mechanisms, leaving policy to the users of the system. The “what” or design issues are perhaps more difficult than the “how” or implementation/automation issues.

Views of Collaborative Systems

As the above discussion shows, can decompose the area of collaborative systems into design, implementation, and infrastructures. We can study this area from several other, overlapping views:

- *Real-World Problems*: What collaboration problems can collaborative applications address, and what kinds of features are needed to solve these problems? This view ensures we are building systems that solve real problems and allows us to develop specialized solution for a problem.

- *Issues* What are the technical issues that must be resolved by these systems and in what ways? From an academic point of view, this is perhaps the ''purest'' approach in that it eliminates repetition of issues common to many domains, and allows us go in-depth into each of these issues. It corresponds, for instance, to the teaching of operating systems by identifying the orthogonal issues such as process management, memory management, and process coordination; and teaching each issue in-turn.

- *Systems*: Which collaboration systems (applications and infrastructures) have been built and what are their properties? Most systems are collections of carefully integrated features, and this view allows us to understand specific collaboration constructs in the context of complete systems. It corresponds, for instance, to the teaching of programming language constructs using the comparative programming language approach, that is, teaching and comparing a variety of complete programming languages such as ML, Smalltalk, and Prolog.

- *Disciplines*: Which existing CS areas/disciplines do they extend and in what ways? This view allows us to understand new collaboration constructs in relation to existing concepts, thereby ensuring that we build on the knowledge and insight developed by traditional fields.

We can use these views to now define the nature and scope of this course.

Driving Problems

The real-world problems addressed by collaborative systems come from several areas:

- *Document Preparation*: Documents of all kinds - papers, proposals, brochures, etc - are often coauthored, and manually managing a coauthoring process, especially when the coauthors are distributed, is generally difficult. Collaborative applications can allow distributed co-authors to easily observe and comment on each others' activities and help ensure the consistency of the document.

- *Business Management:* Businesses require teamwork to be successful and are often distributed. Collaborative applications can enable managers in distributed businesses to make better group decisions, follow business processes, and monitor the status of ongoing projects.

- *Software Engineering*: Every phase of software engineering - requirements analysis, design, coding, testing, and maintenance, requires collaboration, and software development is rapidly getting distributed. Collaborative applications can help distributed software engineers share the results of software development tools such as debugging and testing tools, and to follow software processes.

- *General Engineering*: The benefits of collaborative software engineering extend to the general area of computer-aided engineering design/manufacturing, thereby allowing what is known as

''just-in-time'' engineering. Of particular interest here are distributed collaborative VR interfaces for simulating real-world objects being engineered such as ships/automobiles.

- *Science*: The National Science Foundation has developed the vision of distributed collaboratories, allowing geographically-dispersed scientists working together on national/international projects to exchange results in a timely fashion and monitor and discuss data gathered from remote instruments.

- *Art:* People might collaborate in some artistic endeavor, such as music composition, creation of a photo album, and movie editing.

- *Education*: There is increasing interest in distance education, especially in sparsely populated areas. Collaborative applications can enable both synchronous and asynchronous distance education by allowing teachers to lecture to students in remote sites, lab assistants to consult with remote students, and distributed students to collaborate with each others.

- *Medicine*: Collaborative applications can enable tele-medicine, and as mentioned in the discussion of tele-presence, possibly even tele-surgery; improve the communication among medical personnel; and ensure that proper medical processes are followed.

- *Air Traffic Control*: One of the trickiest coordination problems is air-traffic control. Collaborative applications can allow air-traffic controllers and pilots to view up-to date status information regarding the positions of aircrafts.

- *Command and Control*: Similar problems arise in military command and control operations, and collaborative applications can support planning and monitoring of these operations.

- *Surroundings awareness:* People might exchange traffic, weather and other kinds of data to gain awareness of their surroundings, which in turn can help them, for instance or reduce traffic congestion, determine if they should visit or evacuate from some other location.

- *Universal access:* This goal of this field to allow people with all kinds of abilities access to computing. In the context of collaboration, this would mean allowing users with different abilities to collaborate with each other using the computer.

- *Games/Social Interaction*: Collaborative systems do not support only work – they can include other forms of interaction such as playing games and, perhaps more importantly, interacting in virtual meeting places or social networks

The above discussion shows that there is a large variety of driving problems for work in collaborative systems. A more abstract and simpler motivation that any complex problem requires collaboration, and today such collaboration tends to be distributed.

Issues

Collaborative applications raise several design and implementation issues. The former address the user-interface of the application and correspond to the collaboration functions provided by the application to end user. The latter address how the application is programmed and its performance.

The design issues include:

- *Single-user interface*: What is the single-user interface presented to a user, that is, what are the application semantics if there is a single user in the session?
- *Session Management*: How do distributed users create, destroy, join, and leave collaborative sessions?
- *Coupling*: In a multiuser session, what feedback do user receive in response to the input of another user and when do they receive it?
- *Access Control*: How do we ensure that users do not execute unauthorized commands?
- *Concurrency Control*: How do we ensure that concurrent users do not enter inconsistent commands?
- *Merging*: How do we merge concurrent commands entered by different users?
- *Process control*: How do we ensure that users follow prescribed group processes?
- *Undo/Redo*: What are the semantics of undo/redo in a collaborative session?
- *User Awareness*: How are users made aware of ``out of band'' activities of their collaborators, that is, activities not deducible from the application feedback they receive from coupling?

The implementation issues include:

- *Design Patterns*: What kind of objects are used to program collaborative applications, that is, what are the design patterns for implementing these applications?
- *Collaboration Awareness*: Which of these objects are collaboration aware and how are these objects integrated with existing, collaboration-unaware objects?
- *Layers*: How are these objects stacked in software layers?
- *Concurrency*: How is the application decomposed into concurrent threads executing on the same or different computers?
- *Distribution*: How are the application objects and threads placed on the computers of the collaborators and other, special-purpose, servers?

- *Replication/Migration:* Which of these objects are centralized on a single computer and which are replicated on all computers involved in the collaboration? Which of the centralized objects can migrate?

- *Display Consistency*: How do we ensure that a series of potentially non-commuting user actions shared at two sites are effectively executed in the same order?

- *Real-Time Support:* What kinds of services are provided to ensure real-time interaction with tolerable jitter and latency? Jitter occurs in distributed applications when messages are received and processed at a different rate from which they are sent.

- *Infrastructure Support*: Which of the application objects are implemented by the application programmer and which are provided by an infrastructure.

- *Interoperability*: How are objects of a collaborative application integrated with objects of other (collaborative and non-collaborative) applications?

Systems Disciplines

We have seen earlier that a variety of disciplines are closely related to collaborative applications and infrastructures. Now that we have an idea about the issues, let us revisit some of these areas and identify some specific concepts of relevance to collaboration systems. We will look only at some of the "systems" areas, that is, those that provide some kind of infrastructures, as these areas tend to embody general concepts. We will look at concepts in them related to collaboration infrastructures:

- *Operating Systems:* The goals of an operating system are to manage objects shared among multiple, possibly distributed processes. As users collaborate with each other through processes that share objects – an infrastructure that manages these objects, then, becomes part of the operating system. Current operating systems allow the construction of a wide variety of collaborative applications and address some of the issues mentioned above such as access and concurrency control. However, they have not been designed to support applications supporting close collaboration, and as a result offer only low-level support for building these applications. Therefore, collaboration infrastructures are layered on top of traditional operating systems as middle ware, though its goals are consistent with those of the underlying operating systems.

- *Database Management Systems:* Like traditional operating systems, traditional database management systems provide automatic storage and access of data shared among, potentially distributed, processes executing on behalf of different users – facilities collaboration infrastructures must also support. Like operating systems, database management systems support access and concurrency control, but at the granularity of table elements rather than files. In addition, they support recovery, which is related to multiuser undo. What they lack are facilities for defining complex data that goes beyond tables and relaxation of the traditional transaction model to support a variety of couplings.

- *Programming Languages:* The goal of a programming language is to provide non-shared abstractions for supporting the construction of software applications. Collaboration infrastructures often convert these abstractions to shared abstractions.

- *User Interface Systems:* These systems provide abstractions for performing single-user I/O. We will study how these systems can be extended to support multiuser I/O. Important questions here are: At what level of a user interface system must multiuser I/O be supported: at the window system, toolkit, or the application level? How should the abstractions at this level be changed?

- *Software Engineering:* Many of the techniques and tools developed for supporting software engineering are relevant for collaborative infrastructures including tools for supporting process control and interoperability. (As we see later, software engineering is also an important driving problem for collaborative applications.)

Research and Industrial-Strength Systems

Several important systems, both applications and infrastructures, and research and industrial-strength, have been developed to support collaboration.

Research Applications

Let us first consider some of the pioneering research applications:

- RTCAL (Real-Time Calendaring System): Developed as part of a Ph.D. thesis at MIT, it allows multiple users to schedule meetings in real-time. This research was instrumental in identifying several of the implementation issues in collaborative systems.

- Cognoter: Developed as part of the Xerox Colab suite of collaborative applications, Cognoter allows users to collaboratively brainstorm the structure of a presentation or paper. A unique feature of this system is an automatically enforced meeting process. This research also helped identify issues in coupling among users, experimenting both with WYSIWIS (What You See Is What I See) and non-WYSIWIS coupling.

- CES (Collaborative Editing System): Developed at MIT, this collaborative document editor supports time-based implicit commitment of transactions, mentioned earlier.

- Grove: Grove is a group outline editor, developed at MCC. It recognizes the structure of the outline, provides fine-grained access control and non-WYSIWIS user interface, and relies on social protocols for concurrency control.

- ShrEdit: Developed at the University of Michigan, this is a multi-user text editor that also supports non-WYSIWIS coupling. In such interfaces, users cannot see what others are viewing. Therefore, the tool allows each user to also see the private view of each of the collaborators, which does not scale well to a large number of users.

- GroupDraw : GroupDraw is a group drawing tool, developed at the University of Calgary. It addresses the scalability problem of the ShrEdit approach by providing various forms of summarization of the view of collaborators. In addition, it provides a new form of concurrency control called optimistic locking.

- Quilt: Developed at Bellcore, provides rich support for collaborative asynchronous composition of a document, including logging of concrete and abstract user actions, typed annotations, and role-based access control.

- PREP (work in PREParation editor): Also supporting asynchronous collaborative writing, PREP has been developed at CMU. It provides a column-based interface for commenting, flexibly pinpointing of the differences between different versions of the document, and epidemic algorithms for synchronizing edits by different users.

- Information Lens: Developed at MIT, Information Lens invented the idea of semi-structured email messages.

- Coordinator: Developed at Stanford, it structures collaboration using a theory of collaboration called the ''speech-act'' theory for characterizing the nature of collaborations. It can be considered one of the first workflow systems.

- IBIS (Issue Based Information System): Developed at MCC, this tool structures the discussion of an issue.

- Hydra: Developed at the University of Toronto, the Hydra audio/video conferencing system allows each remote collaborator to be assigned a different region of the local physical space.

- MUD (MultiUser Dungeons): Developed by researchers at Xerox and elsewhere, MUDs provides a text-based persistent virtual environment in which people can meet and interact with each other based on which rooms they have entered. Because it was persistent, it was used to communicate a variety of virtual communities such as an astronomy community.

- DIVE (Distributed Interactive Virtual Environment)/MASSIVE: Developed about the same time at Amsterdam and University of Nottingham, DIVE and MASSIVE, like MUDs, provide virtual meeting environments except that these environments are based on 3-D graphics.

Industrial-Strength Applications

Today, we also see a variety of industrial-strength applications, many of which build on the research applications mentioned above:

- SharePoint, Microsoft Groove Folders: These provide shared document repositories. SharePoint centralizes the documents at a server, while Groove replicates them on the computers of the users sharing the documents.

- Microsoft Word: Microsoft Word offers a variety of collaboration functions. Like PREP, it supports annotations, and like PREP, it provides fine-grained merging and diffing. In addition, it provides powerful facilities for tracking changes of different users.
- Microsoft OneNote and Google Docs: Microsoft Word does not offer real-time collaboration of the kind provided by ShrEdit, Grove and GroupDraw. One reason is that it is almost impossible to understand and change the huge amount legacy code in it. Newer and lighter weight document editors, such as Microsoft OneNote and the Writely document editor (now part of Google Docs), do offer real-time collaboration; as does the Google Docs spreadsheet editor.
- Instant Messengers and Social Networks: These are extensions of the basic idea in MUDs of creating a text-based distributed virtual community.
- Second Life: Similarly Second-Life and other distributed virtual environments we see today are extensions of the idea in Massive and Dive of 3-D shared environments.

Research Infrastructures

Hand-in-hand with the research and development of collaborative applications has gone the research of infrastructures for automating these applications. Again, we consider the most important systems:

- NLS: Douglas Englebert, a Turing award winner and the inventor of the mouse, created the first collaboration infrastructure, called NLS, as early as 1968. It was a system that allowed a user to share the screen of another user. Englebert did this on his own, after he left Xerox because his ideas were considered too radical even for this very innovative company. As we see below, it took Xerox twenty years to catch up with this work!

- Colab Programming Environment: Developed as part of the Xerox Colab project, this environment extends an object-oriented programming language with constructs for providing a fixed policy for sharing arbitrary objects among users. It replicates the shared object on the computers of all users, much like Yahoo replicates email on multiple machines of a server farm. It was used to develop the Cognoter and other applications in the Xerox suite of collaborative applications mentioned above.

- VConf, Rapport, XTV: The Colab programming supported collaboration-aware applications. Developed at Stanford, Bell Labs, and UNC, respectively, VConf, Rapport, and XTV are extensions of window systems, supporting sharing of existing collaboration-unaware applications developed using the extended window systems. They differ mainly in the choice of the underlying window system. Each of these systems was implemented using both the replicated and centralized architecture.

- Rendezvous and Weasel: Being extensions of window systems, the systems above forced users to share the complete contents of an application window. Bellcore's Rendezvous provides a system allowing application-developers to define spreadsheet-like constraints between shared data and the views different users have of these data. It was used to develop a card game that allowed players to see different views of the game state. Weasel, developed at York University, also supports such constraints, but provides a special caching scheme to ensure that they are

evaluated efficiently when the data and the views are on different computers. Unlike Xerox's Colab programming environment, both assume that the shared data are centralized on a single computer.

- Suite, developed at Purdue and UNC, also assumes centralization of shared data. However, it takes a different approach to allowing customization of the collaboration semantics. Instead of defining constraints, programmers set values of a set of predefined collaboration parameters defining the coupling, undo, concurrency control, merging, and access-control policies. These parameters cover all collaboration policies implemented so far. In particular, they allow a single system to support database transactions, email, IM, synchronous single-view editing, synchronous multi-view editing, and asynchronous/disconnected editing. Suite is integrated with the C programming language in that it assumed the shared data were described using C types. It provided fine-grained coupling, merging, locking, and access control of data structures defined using these types. For example, the programmers can specify which users can access a record field.

- GroupKit: C does not support tables, a powerful construct, as users of scripting languages know, for defining a variety of data structures. GroupKit builds on this insight by supporting synchronous sharing of tables. Like Xerox's Colab environment, replicates shared objects on the computers of all users. In addition, it provides special widgets for supporting awareness, which were used and evaluated as part of the GroupDraw application mentioned above.

- Sync: Sync, developed at UNC, combines the goal in (a) Xerox's Colab programming environment of sharing arbitrary encapsulating objects, and (b) Suite of providing fine-grained sharing policies. Unfortunately, these goals are conflicting as an encapsulated object hides its structure and thus cannot be decomposed to define fine-grained policies. Sync resolves this problem by assuming certain conventions or programming patterns for exposing the logical, but not physical, structure of an object. Sync supports both connected and disconnected editing of shared objects, and provides merging of different versions of these objects. One of the Sync applications is a set of replicated folders that are automatically synchronized.

- Logger: As we saw above, some system replicate shared objects while other centralizes them. The logger system, developed at UNC, allows this decision to be changed dynamically for an application. It also provides special scheduling and multicast algorithms for enhancing response times.

- GT: Developed at UNC, this system, like the Logger has special algorithms for ensuring efficient communication of messages in a collaborative application.

Multiple Views of Same Concept

We have seen above four different views of collaboration systems: driving problems, issues, disciplines, and systems (applications and infrastructures). These views are overlapping – in it is possible to understand the same concept from all of these views. To illustrate, consider the concept of WYWIWIS interaction. Figure 21 shows all four views being taken to understand it: (1) Problem: It is necessary to

support distributed pair programming in collaborative software engineering. (2) It is one approach to address the coupling issue. (3) Discipline: Like database transactions, it determines how users actions are shared. (4) Systems: It is supported by the VNC system.

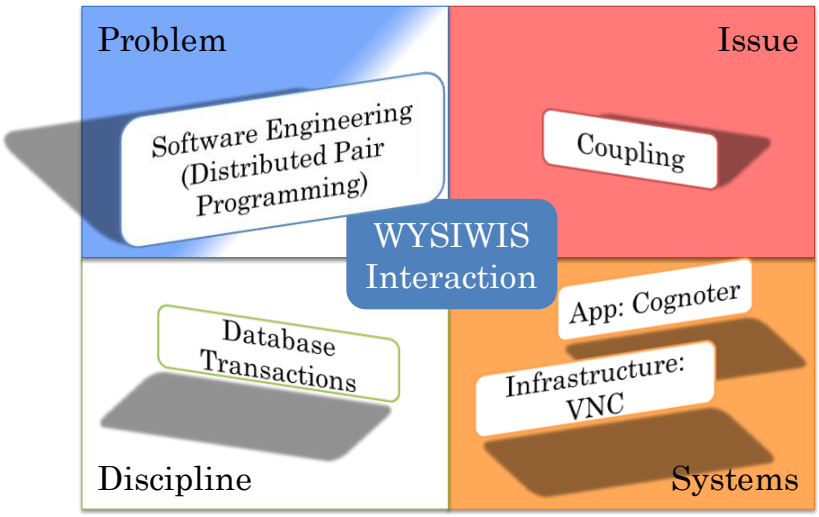

Figure 15 Multiple Views of WYSWIS

Thus, the four views define a network of related concepts.

In this overview chapter, we have superficially traversed the most important concepts in this network from all four points of view. We will not be able to so as we study the subject in more depth to meet the time constraints and avoid repetition. For instance, we will not consider all the possible domains in which WYSWIS coupling could be useful and all the systems that have implemented it, as doing so will probably result in tedious repetition and take substantial time. However, when necessary we will explain a concept from multiple views.

We will take the issue, system, and problem views to various degrees. We will consider all of the issues mentioned here. In addition, we will consider in depth a few infrastructures and a variety of applications. These systems will be explored both in class lectures and home-works. We will also study in depth the problem of collaborative software engineering, as this domain is of particular interest to computer scientists.

Even though we will not use all views completely as primary top-level views, we will, nonetheless use them as secondary views. That is, for each feature we introduce as part of a system or as a resolution to an issue, we will consider the problems it solves and the relevant research in related disciplines. Since we will be looking at overlapping sets of concepts from two different views, there will be some repetition. The systems view will give us the broad context for many of the collaboration constructs we will discuss in-depth when we take the issue view later.