

An Active Hypertext Model for System Requirements

Colin Potts
College of Computing
Georgia Institute of Technology

Kenji Takahashi
NTT Software Laboratories
Nippon Telegraph and Telephone Corporation

Abstract

We are developing tools to support a conversational metaphor for requirements definition and analysis. Our conversational model consists of three components: (1) a hypertextual representation of requirements and their interrelations, (2) an issue-based speech act model, and (3) a typology of changes. These components act together in a model we call the Inquiry Cycle. We discuss requirements analysis activities supported by the conversational model, including information retrieval and navigation, rationale management, and agenda management. We have implemented a prototype active hypertext system, and we have applied our model and implementation to the requirements for an ATM banking system, an example we use in the paper for illustration.

1. Introduction

We are developing tools to support a conversational metaphor for requirements definition and analysis. We view requirements definition as an extended dialogue between stakeholders who are planning the capabilities of a new system. Our emphasis lies firmly in inquiry and questioning during the production process, not the linguistic details of the product or the transformational properties of the production process. We pay particular emphasis to where information needs to come from and when.

We have developed a conversational framework, the inquiry cycle, that underpins our technology development. To support the inquiry cycle, we have developed the concept of active hypertext. Our hypertexts consist of interconnected fragments of semi-structured information (usually text) about the requirements. They are active in three senses: (1) they capture the process by which the requirements become elaborated and agreed upon, and not just a snapshot of the current or final state of the requirements, (2) they directly support the ongoing process of inquiry and scrutiny so that users of the hypertext know what information needs to be sought and what decisions or assumptions are pending, and (3) they support information consolidation, so that information of only transitory interest can be simplified and summarized once a decision has been made.

The conversational metaphor, the inquiry cycle and active hypertext technology are our responses to the difficulties faced in requirements definition situations for real computer-based systems. Lubars, Potts and Richter (1993a) in their field study of requirements practices in 23 project organizations discovered that the principal problems faced by project teams were communication, agreement and change.

The elaboration of requirements is an inquiry-driven iterative process. Particularly in the case of market-driven projects, definition and analysis, expression and validation, requirements and design are tightly interwoven.

The next section describes our conversational model. It consists of three components: (1) a hypertextual representation of requirements and their interrelations, (2) a speech act model, and (3) a typology of changes. These components act together in a model we name the *Inquiry Cycle*. Section 3 discusses requirements analysis activities that must be supported by the conversational model. These include information retrieval and navigation, rationale management, and agenda management. Requirements validation, and the specific analyses that help in validation, are regarded as special forms of information retrieval. Throughout, we illustrate our points by referring to the requirements for an ATM banking system. Section 4 describes the current status of our work, particularly the prototype active hypertext system. Section 5 discusses related and previous work, and Section 6 contains our plans for future work.

2. Conversational Model

A conversation consists of four things: participants, shared information, speech acts, and effects. In the case of system requirements, the participants are 'stakeholders', those with a stake in the outcome. We leave this term deliberately vague in preference to defining conversational roles such as 'customer', 'requirements analyst', 'end user', etc. Although the prototype active hypertext system to be described later lends itself to use primarily by analysts working in customer-specific projects with an originating requirements document to start from, other implementations of the conversational metaphor could equally support participatory design.

The remaining three components and their interrelationships are depicted in Figure 1. In the next

three sections, we discuss these three components in turn: shared information (Section 2.1), the speech act model (Section 2.2), and requirements evolution (Section 2.3). In Section 2.4, we then put the components together and present our conversational model, the inquiry cycle.

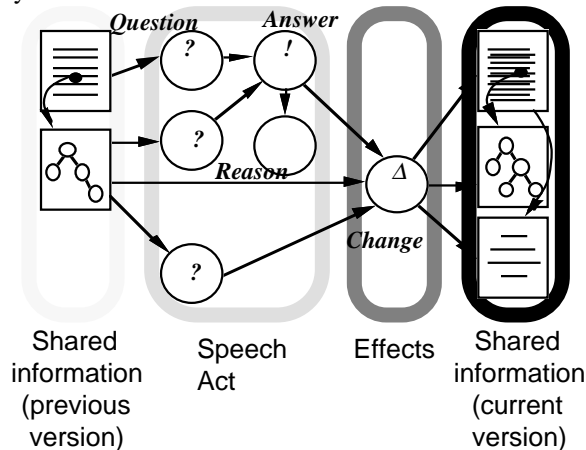


Figure 1. Conversational Model

2.1 Shared Information

Requirements-related information includes requirements, domain-specific facts and assumptions, project planning commitments, implementation constraints, and so on. We assume that this information is represented in natural language text and free-form diagrams. We impose no requirements on and make no assumptions about the granularity or content of these pieces of information, and we call them ‘requirements’ for simplicity.

Granularity There is no externally definable ‘correct’ level of requirements granularity. A collection of requirements should probably be merged as a single requirement if the same questions get asked about each requirement and the answers are always mutually determined. A requirement should probably be split into several requirements if it contains several orthogonal statements; that is, statements that lead to different questions which can be answered independently. These decisions about requirements granularity can only be made as the questions are asked, questions which are generally application-specific and unpredictable in advance.

Requirements often occur in related sets. For example, one requirement may be a generalization, while other requirements are elaborations of it. For our purposes, we consider such cases to be examples of sets of separate — albeit interrelated — requirements, not examples of single, structured requirements.

Content We treat requirements as uninterpreted text or diagrams so that no assumptions are embedded about specification languages or expressive style.

Links Requirements may be interrelated by ‘related-to’ and ‘subsumes’ links. In a linear requirements document, the subsumption structure is usually reflected in the numbering of paragraphs and sections. The ‘related-to’ link denotes a link between requirements that cuts across the subsumption structure. Once again, we could posit a number of subtypes of ‘related-to’ link, each with its own semantics, but we avoid building linguistic assumptions into the model because these would bias the model toward certain types of systems or requirements situations. Historical links, such as ‘refines’ or ‘supersedes’ are derived through the inquiry model and are discussed later.

Other links that can exist between requirements are ‘next’ and ‘previous’, which are used in linearizing the requirements (see Section 3.1).

2.2 Speech Acts

A requirements ‘conversation’ consists of speech acts (Austin, 1962) performed by the participants. For simplicity, we assume in the remainder of the paper that the speech acts are all performed by a single analyst concerning a set of requirements.

In practice, however, several stakeholders may be involved.

We model conversations by a simple speech act model consisting of three components: questions, answers, and reasons. In its structure and purpose, the model resembles the IBIS (Kunz and Rittel, 1970), QOC (MacLean et al., 1991), and Potts and Bruns (1988; Potts, 1989) models.

Questioning is the main speech act in the conversational model and is the departure for the inquiry process. Requirements definition and analysis processes are inextricably interwoven: stakeholders repeatedly scrutinize the requirements by asking questions or raising issues. *Answers* describe solutions to problems raised in questions, candidate refinements or revisions that respond to the challenges posed by the questions. An answer generally answers a single question, but there may be many answers to one question. A *reason* is the rationale for choosing one answer over alternatives.

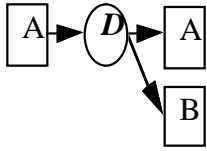
2.3 Requirements Evolution

The ultimate effect of an analysis of requirements is a commitment. The commitment may be to freeze the requirements and act on them (i.e. implement them). More usually, it is a decision to change something.

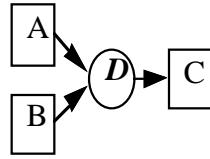
We identify several types of change: clarification, retraction, refinement, splitting, and merging. These

types are depicted schematically in Figure

a. Refinement



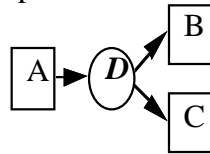
d. Merge



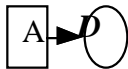
b. Clarification



e. Split



c. Retraction



2. Figure 2. Type of Changes

A *clarification* results in the rewording of one or more requirements. There are no changes to the structural relationships among requirements.

The following example of a requirement demonstrates a clarification:

Before:

“ATMs communicate with the Consortium Information System, which clears transactions with the appropriate banks”

After:

“ATMs communicate with the Consortium Information System, which dispatches transactions to the appropriate banks and is informed of the results.”

Retraction involves the deletion of a requirement from the shared information space. It affects structural relationships in two senses: any ‘related-to’ links connecting the retracted requirement and other requirements or ‘subsumes’ links into the retracted requirement are also deleted, because we require links to connect exactly two requirements. If a retracted requirement subsumes other requirements, those requirements are also retracted.

Retraction often involves narrowing the scope of the automation in the system from the original goals.

A *refinement* results in more complex structural alterations. One or more additional requirements are added that are derived from the refined requirement. Generally the set of resulting requirements are related. However, we do not insist that the original requirements subsume the new ones.

As an example of refinement, consider the requirement to bill banks for using the ATM system:

Before:

“The cost of running the shared system will be apportioned to the banks according to the number of customers with cash cards.”

After:

1) “The cost of running the shared system will be apportioned to the banks according to the number of customers with cash cards.”

2) “The number of cards issued by a bank is reported by the bank to the consortium and the report is entered by an operator.”

Splitting and *merging* are the duals of each other. They occur when the granularity of the requirements is inappropriate. In the case of splitting, the requirement contains orthogonal parts; in the case of merging, the same requirement is stated in different terms in different places. Splitting yields a set of related requirements. Sometimes the resulting requirements are related by subsumption.

Merging is necessary when parts of the requirements document are produced independently. Whenever several authors are involved it is likely that the same requirement will be expressed more than once in slightly different language. Many examples of this phenomenon are reported by Lubars, Potts and Richter (1993b) in their analysis of the requirements for the Tomahawk Weapons Control System.

2.4 The Inquiry Cycle

The components of the model are independent and could be supported independently: the shared information on its own provides a requirements hypertext model and could be supported by a hypertext system, the speech act component on its own provides a decision support model similar to IBIS (Kunz and Rittel, 1970) and could be supported by a tool akin to gIBIS (Conklin and Begeman, 1989), and the evolution component on its own provides a model of requirements change and could be supported by a history management system.

It is the integration of the components that distinguishes the conversational model. We do so in terms of a three-part cycle, called the inquiry cycle. Starting at the top of Figure 3, existing requirements are challenged through the asking of questions. The discussion (the speech act component) may give rise to proposed changes (the evolution component), which when effected, give rise to a new version of the set of requirements.

The relationship between the shared information component and the speech act component means that requirements expression and analysis can be handled within a uniform framework. Requirements and questions about them, proposed answers, reasons for choosing one formulation (for example, a prioritization) over another are represented in one connected web of semi-structured information fragments.

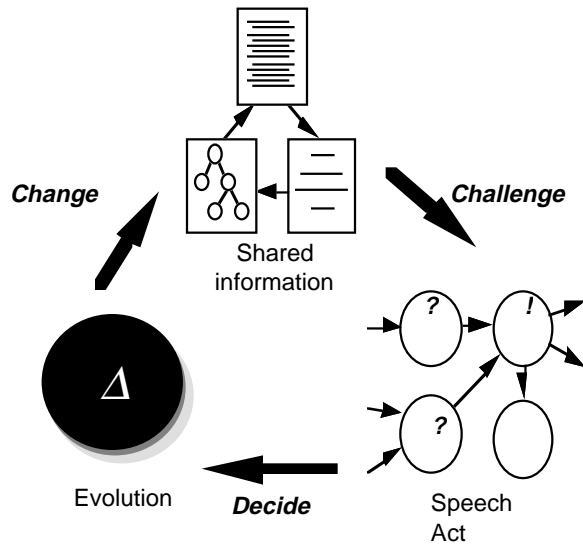


Figure 3. Inquiry Cycle

Similarly, the connection between the shared information and evolution components means that it is possible to return to versions of the requirements that have been superseded. Changes are objects in their own right that sit between versions of requirements.

Finally, the connection between the speech act and evolution components means that it is possible to tie discussions about requirements to the effects of these discussions. From the perspective of a change that has taken place, it is possible to go back to the discussion that gave rise to it (suitably consolidated — see Section 3.2) and use that to reconstruct the rationale for the decision to refine or include a requirement.

The purpose of the inquiry cycle is to capture the possible range of changes to requirements as they evolve. It is certainly possible to bypass some of the components some or even most of the time. For example, a requirement may simply be changed with only the most cursory discussion or without any discussion at all. Even the change itself may be ignored as a first-class object.

Project-specific policies can be imposed on top of the conversational model that, for example, prevent certain classes of stakeholder from making certain changes or preventing changes to certain requirements. We do not build these policies into the framework itself.

Figure 4 shows the evolution of a small set of requirements for the ATM system.

3. Requirements Analysis Activities

Three primary activities are considered in this section: (1) information retrieval and navigation, (2) rationale management, and (3) agenda management. Authoring, the creation and revision of objects of the types discussed in Section 2 (requirements, questions, etc.) is a trivial activity and is not discussed further.

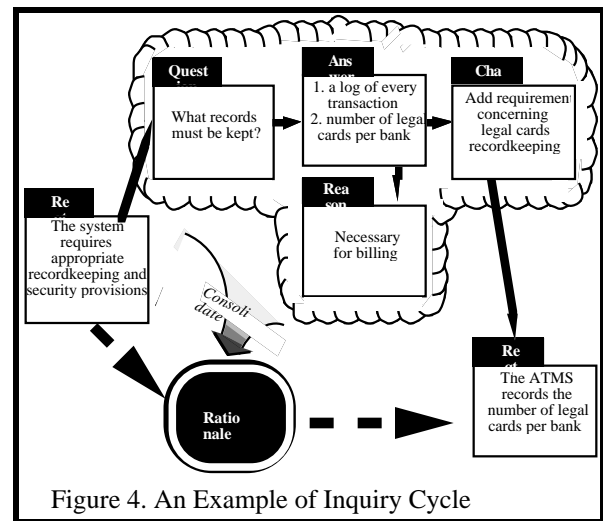


Figure 4. An Example of Inquiry Cycle

3.1 Navigation

Navigation refers to the activity of reading or finding requirements. Simple navigation involves reading individual requirements and moving from related requirements to requirements. Another form of navigation is querying: formulating a general query and obtaining a list of matching requirements.

The model supports these forms of navigation directly. Requirements can be found by following links from other requirements either directly or by formulating structural queries. Queries that refer to the contents of requirements, for example by referring to contained keywords or by syntactic analysis, are also possible but not directly supported by the model.

A more elaborate form of navigation is linearization. A linear requirements document is a 'report' derived from the requirements hypertext (Potts, 1984) by executing a standard search procedure.

3.2 Rationale management

Requirements change rapidly during requirements definition. Often, stakeholders forget why a requirement has been decided upon or eradicated. To avoid unnecessary repetition and discussion, it is important to record the rationale for significant decisions. It is equally important to be flexible in the degree of detail required in the rationale record. Early in the requirements definition process, for example, it is important not to record the reasons for every small change, otherwise stakeholders will get swamped by volumes of irrelevant or obvious information and those who write the requirements will spend more time justifying every minor revision than analyzing requirements. In contrast, later on in a formal project it may be necessary to insist on reasons for every change, no matter how small.

Rationale can sometimes be articulated by the author of the requirement. Sometimes it is too much trouble to bother to record reasons for a small change, or the reason is obvious — either because the rationale is contained in the content of the requirement and does not need to be recorded separately or is implicit in the mode of derivation. There are two ways to handle rationale: explicitly recording local rationale (see Section 2.2), and consolidating rationale by using content or structural information.

Rationale consolidation follows a standard pattern. It involves collapsing a structural discussion into a single form that summarizes the discussion and therefore the rationale for the ensuing change. Simple natural language generation is sufficient to give informative rationale, provided that the type of change that ensued is known. For example, if the change is a refinement, the rationale will be stated differently from the case when the change is a clarification.

The rationale for the change in Figure 4 is given in Figure 5. The requirements enclosed in angle brackets are anchors to hypertext links.

Refine <Reqt> by <Reqt>
to answer "what records must be kept?"
and decide "1. a log of every transaction,
2. number of legal cards of bank"
because "necessary for billing"

Figure 5. Consolidation of Rationale

There are two different ways to find out the nature of the change: (1) the stakeholder may declare the change to be of a certain type; (2) the type of change may be inferred from the textual differences between requirements versions and the topology of the hypertext.

3.3 Agenda Management

The conversational metaphor supports requirements definition and analysis as activities that are scheduled in time. Because requirements unfold over time as new information is acquired, requirements definition and analysis activities can be planned and scheduled. This function is known as agenda management.

Agenda management refers to the monitoring of ongoing work and the scheduling of completion tasks. These tasks may be performed by any stakeholders involved in the requirements process, and a specific task will usually be the responsibility of certain stakeholders. To avoid embedding assumptions about the project organization, we do not build in rules about the types of action and the roles responsible for them. These rules would be implemented by project-specific policies or task specific decisions.

Since the inquiry cycle defines a cyclic process of definition, inquiry and evolution, the agenda management function supports the completion status of activities at all three points of the cycle. In the shared information component, agenda items include monitoring the inquiry status of current requirements, checking whether requirements are satisfactorily linked and checking the completeness of method-specific analyses. The timing of these analyses is specific to the method or project.

Agenda management in the speech act component refers to the completion status of discussions about requirements. It is useful to ask which questions have been posed and then appear to have been forgotten. It is also useful to find out which questions have been asked rhetorically but need to be asked of a specific stakeholder or information source before an answer will be obtained. This is useful in setting the agenda for a meeting with a customer. Similarly, it is useful to have a list of answers that have not been acted upon so that proposed changes can be discussed.

Finally, in the evolution component, the agenda management function examines the completion status and plans the completion of proposed changes. A change may be proposed (and discussed) long before it is enacted. It is important, however, to know which committed changes have not been made so that they are not forgotten. This type of agenda management is especially important in larger customer-specific projects in which changes are the responsibility of a change control board.

We have implemented a

4. Current Status prototype active hypertext system using *Hyperbole* (Weiner, 1992). The prototype supports the three components of the inquiry cycle model: linked requirements, speech acts and evolution. The prototype supports navigation and traceability activities, agenda management, and rationale management and consolidation.

Hyperbole is a hypertext system built on Emacs, and is easily customized and extended. In the prototype, the hypertext nodes (requirements, speech acts and changes) are stored as separate files. Only text files are supported.

'Buttons' are embedded in the nodes and are presented to the user as text between angle brackets (e.g. Figure 5). A button may be selected whenever it is visible. In the prototype, buttons appear in two rows at the foot of each node: one row contains 'find' buttons, buttons that are used to navigate through the shared information, speech acts and changes; the other row contains 'do' buttons, buttons that perform some operation that changes the state of the network.

4.1 'Find' Buttons

We provide a full range of 'find' buttons, which implement navigation operations from every type of object to related objects of the same and other types. For example, from a requirement, it is possible to navigate

to the next requirement in the linear sequence, the consolidated rationale for the requirement, questions about the requirement, and the previous and next versions of the requirement.

Many navigation operation could lead to many nodes, not just one. In these cases, our prototype presents a menu for the user to select from. The menu is also implemented as a set of buttons.

4.2 'Do' Buttons

We provide buttons for a number of operations specific to each object type. For example, from a requirement, it is possible to raise a question or clarify the requirement in a single operation. Raising a question causes a question node to be created and linked to the requirement. Clarifying a requirement, causes a new version of the requirement to be created with the same content. The user may then edit the content. When the new requirement is saved, the old version is marked as superseded and a change node is created to mediate the clarification from old to new version.

The implemented buttons allow the user to bypass the inquiry cycle if desired. For example, it is possible to change a requirement directly without initiating a discussion.

4.3 Rationale Consolidation

We are currently implementing rationale consolidation. A tool, *rdiff*, is instrumental in generating the rationale for sets of requirements from the rationale of constituent requirements and their attached speech acts. In one mode, *rdiff* explains the difference between the current version of a requirement or set of requirements and the most recently approved version of the requirement. Approved requirements may be justified in terms of customer authority (i.e. their justification is that the customer says so) or they may be justified by further consolidation.

Rdiff is an extension of the the Unix command *diff* in the spirit of Neuwirth et al's (1992) "Flexible Diff". It automatically finds the previous version of each requirement by tracing back through changes and generating the rationale for the requirements in the form of Figure 5. In *rdiff*, we distinguish between types of change (see Section 2.3) and low-level editing operations that implement the changes.

Rdiff distinguishes four types of editing operation: append, change, delete, and identity (no change). The current algorithm uses only the text of the requirements being compared. We believe that the derivation history (i.e. the topology of the hypertext) can be used to recover the high-level changes that gave rise to the later version. We are working on and implementation of some change recovery heuristics.

4.4 Agenda Management

Agenda management is being implemented for the queries described in Section 3.3.

4.5 Worked example

We have completed a worked example based on Rumbaugh et al's (1991) specification of an ATM system. We are currently working on a more detailed example, a group meeting scheduler (Van Lamsweerde, personal communication).

The inquiry cycle model integrates several views of requirements and design elaboration that have received attention from other authors.

The speech act component is derived from IBIS (Kunz and Rittel, 1970). Other authors who have explored the role of issue-based representations for requirements and design rationale are Lee (1991) and Arango et al. (1991). Unlike these authors, who have attempted to extend IBIS to increase its representational power, we have made the representation of speech acts as simple as possible. To meet the common criticisms of speech act theories, the rigidity of which has led to mixed success in practical applications (Burgess-Yakemovic and Conklin, 1990; Flores, Graves, Hartfield and Winograd, 1988), we permit relaxation of the rules of the speech act component.

Our evolution component is influenced by the work of researchers in the program transformation community, who have developed more elaborate, language-specific, taxonomies of transformations and tools. For example, Feather (1987) outlines a theory of transformations that do not preserve correctness. Later work in this vein led to the ARIES environment (Johnson, Feather and Harris, 1992). Neuwirth et al (1992) have implemented a flexible diff algorithm for collaborative writing applications.

The inquiry cycle model builds on the work of Potts and Bruns (1988) and Potts (1989). 6. Future Work

So far, we have considered the elements of the shared information component to be unstructured textual paragraphs or diagrams. We intend to extend the notion of shared information to more structured requirements-related information. In particular, we are researching the role of scenarios in the definition and validation of requirements. Scenarios have a well-defined two-dimensional internal structure (Rubin and Goldberg, 1992). One dimension denotes the different participants in the scenario (objects in an object-oriented analysis); the other denotes the temporal sequence of actions or events.

Scenarios are just one example of composite requirements-related documents. We are also exploring

the integration of the conversational model with more traditional CASE documents or diagrams. We are also investigating the interaction between versioning and composite objects.

Once multiple types of information are introduced, we must consider traceability. Traceability is achieved by recording links from originating to derived documents and tracking them when the requirements change.

The forms of traceability that we are investigating are those from customer-generated requirements to derived requirements, requirements to test cases and scenarios, requirements-related information to requirements proper, and informally stated requirements to more formal statements of the same requirement.

The mechanism for establishing and tracking links should be independent of the requirements or design methods used. Specification and design languages and methods set up policies for traceability and what it means for one document to describe the implementation of requirements specified in a predecessor. Thus traceability links may, in some cases, be computed according to formal principles or the derived document can be constructed semi-automatically by means of formal derivation rules.

We have made the claim in several places in this paper that we wish to avoid embedding organizational or project policies in the conversational model. Such policies do exist, however, and must be layered on top of the basic model described above. Among the policies that we are now looking at or plan to support in the future are versioning, method-specific requirements, speech acts and changes, and role differentiation in the requirements process among participants.

Finally, our case study evaluations to date have been realistic, but concocted. We have access to large bodies of requirements documentation for telephone systems, and we intend to apply what we have learned in the small examples to these.

References

- Arango, G., L. Bruneau, J.-F. Cloarec, and A. Feroldi, "A Tool Shell for Tracking Design Decisions", *IEEE Software*, March, 1991, pp. 75-83.
- Austin, J.L., *How to Do Things with Words*, Oxford, Clarendon Press, 1962.
- Burgess-Yakemovic, K.C. and J. Conklin, "Report on a Development Project Use of an Issue-Based Information System", *Proc. CSCW'90*, ACM Press, 1990, pp.105-118.
- Conklin, J. and M. Begeman, "gIBIS: A Tool for all Reasons", *J. Am. Soc. Inf. Sci.*, May, 1989, pp. 200-213.
- Feather, M.S. "Language Support for the Specification and Development of Composite Systems" *ACM Trans. Programming Languages and Systems*, 9(2), 1987, pp. 198-234.
- Flores, Fernando, Michael Graves, Brad Hartfield and Terry Winograd, "Computer Systems and the Design of Organizational Interaction", *ACM Trans. Office Inf. Sys.*, April, 1988, pp. 153-172.
- Johnson, W.L., M.S. Feather and D.R. Harris, "Representation and Presentation of Requirements Knowledge", *IEEE Trans. Software Eng.*, 18(10), 1992, pp. 853-869
- Kunz, W. and H. Rittel, *Issues as Elements of Information Systems*, Working Paper 131, Inst. Urban and Regional Development, Univ. California at Berkeley, 1970.
- Lee, J. "Extending the Potts and Bruns Model for Recording Design Rationale", *Proc. 13th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1991, pp. 114-127.
- Lubars, M., C. Potts and C. Richter, "A Review of the State of Practice in Requirements Modeling", *Proc. RE'93: IEEE Int. Symp. Requirements Eng.*, IEEE Comp. Soc. Press, 1993a, pp. 2-14.
- Lubars, M., C. Potts and C. Richter, "Developing Initial OOA Models", *Proc. 15th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1993b, in press.
- MacLean, A., V. Bellotti, R. Young, and T. Moran, "Reaching through Analogy: A Design Rationale Perspective on Roles of Analogy", *Proc. CHI'91*, ACM Press, 1991, pp. 167-172.
- Neuwirth, C.M., R. Chandhok, D.S. Kaufer, P. Erion, J. Morris and D. Miller, "Flexible Diff-ing in a Collaborative Writing System", *Proc. CSCW'92*, ACM Press, 1992, pp. 147-154.
- Potts, C. "Understanding Complex Descriptions" in G.C. Van Der Veer, M.J.Tauber, T.R.G. Green, and P. Gorny (eds.) *Readings on Cognitive Ergonomics: Mind and Computers*, Springer-Verlag, Lect. Notes in Comp. Sci., 178, 1984, pp. 81-91.
- Potts, C. "A Generic Model for Representing Design Methods", *Proc. 11th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1989, pp. 217-227.
- Potts, C. and G. Bruns, "Recording the Reasons for Design Decisions", *Proc. 10th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1988, pp. 418-427.
- Rubin, K.S. and A. Goldberg, "Object Behavior Analysis", *Comm. ACM.*, 35(9), 1992, pp. 48-62.
- Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.
- Weiner, B. *Hyperbole Manual: Everyday Information Management*, Edition 3.01P, Brown University Technical Report, March 2, 1992