ACH WALKTHROUGH: DESIGNING AND BUILDING A WEB APPLICATION FOR COLLABORATIVE SENSEMAKING

by

Robert Jeffrey Pardoe Wilson

A thesis submitted to
the Faculty of Graduate Studies and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of

MASTER OF COMPUTER SCIENCE

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
September, 2015

Contents

List of Figures

Abstract

This thesis describes the research and development of a prototype for a co-located collaborative intelligence analysis tool: ACH Walkthrough. The tool is a collaborative variation of an established structured analysis method called *Analysis of Competing Hypotheses*, originally developed for intelligence analysis. Recent changes to web application architectures offer important opportunities to produce visually rich applications that support co-located and remote collaborative decision making scenarios. We begin by reviewing the literature on sensemaking and development frameworks for surface applications. We then explore architectural issues in using web frameworks for collaborative applications. We then present the design and implementation of ACH Walkthrough, our prototype design for team-based intelligence analysis, and the evaluation of the application's major architectural components. Finally, we document feedback on the tool and explore alternative designs and architectural approaches.

Acknowledgements

I would like to express my appreciation to my family, who understood how important this mission has been to me, and who helped in no small way to make it possible.

I also wish to thank my friends and colleagues at Carleton's HotSoft Laboratory. It was an honour and a pleasure to work with all of you.

Finally I wish to thank all the instructors and professors at Carleton. Most importantly I wish to express my endless gratitude to Dr. Robert Biddle for taking a chance on an older student.

I would also like to thank the Natural Sciences and Engineering Research Council of Canada, SurfNet: the NSERC Strategic Research Network on Surface Applications, and the Government of Canada.

I dedicate this thesis to my brother Ian,
who showed me how to keep going.

Chapter 1

Introduction

1.1 Motivation

This thesis describes the research and development of a prototype for a co-located collaborative intelligence analysis tool. The prototype is called ACH-Walkthrough (ACH-W, Figure 1.1), a collaborative variation of an established method the structured analysis method *Analysis of Competing Hypotheses* [36] originally developed for intelligence analysis.

The ACH process was developed by Heuer to assist CIA analysts in reasoning about fragmentary and potentially deceptive evidence. He recognized a pervasive risk of cognitive bias demonstrated in the habit of analysts to seek evidence that would confirm their beliefs rather than refute them. This natural reflex could leave analysts vulnerable to deliberate deception, but errors could easily come from completely accidental misdirection as well. Heuer's technique is meant to illuminate unexplored corners of one's reasoning.

Our goal in designing ACH Walkthrough (ACH-W) was to support this reasoning with a web application, and in addition to provide a visual foundation for collaborative analysis by leveraging interactive visualizations on a large display. The resulting prototype was meant to be used in similar intelligence settings, but could also be applied in a number of other sensemaking and decision-making scenarios in both personal and professional settings.

The term *sensemaking* refers to activities involving the collection, categorization, transformation, and interpretation of data for use in decision making. While we may live in an era of unprecedented commercial interest in *artificial* intelligence, there remain certain problems—commercial and otherwise—for which it is still conceivable that only *human* intelligence will suffice. This is particularly true in intelligence analysis, where situations can call for reasoning more sophisticated than we can expect from any kind of automation, and where recommendations for action must still be made despite having only novel, fragmentary, and perhaps even deliberately deceptive evidence available.

Although we know automation falls short in these cases, human minds aren't always the

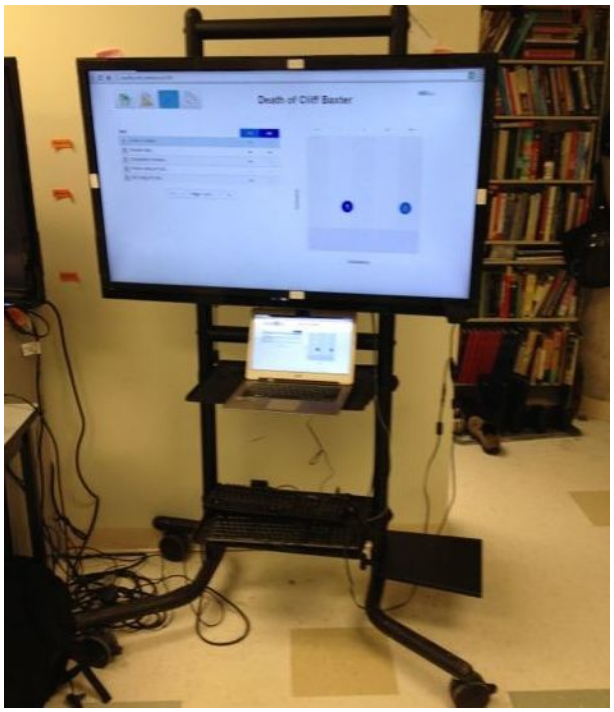


Figure 1.1: Introducing ACH Walkthrough

whole answer either. Even the most prepared minds have cognitive and perceptual limits, such as the number of simultaneous streams of dialogue that can be followed or the number of digits that can be quickly committed to memory. Most people are also susceptible to a variety of perceptual biases, whether they are working alone or in groups. There are procedures and techniques that can counteract or at least partially mitigate our biases. These procedures don't always require technological assistance, but researchers in the challenge of sensemaking have identified "leverage points" [61] where computers might help.

One important challenge in sensemaking is the need to justify the expense of automation, particularly if the work has limited reusability. Since the human cost of design and validation are limiting factors, there is real value in identifying tools that help to rapidly create and rapidly deploy proofs-of-concept and high fidelity prototypes to validate designs within these necessarily secretive procedures.

Wherever people come together to review and discuss information, interactive touch surfaces have the potential to help [11]. Research has also shown that the size of the surface area can offer valuable cognitive benefits. Automation can clearly help with sensemaking, for example through exploratory data visualizations.

Heuer's method seeks to overcome certain well-known problems in reasoning from uncertain evidence. Human brains have evolved to seek patterns, and for much of our evolution the ability to do find the right patterns quickly could mean the difference between life and death. Unfortunately for us, one way our brains do that is to reason from first impressions. This *anchoring* effect [41] might have helped our ancestors think more quickly and thereby avoid physical danger, but it can be a source of error when more nuanced reasoning is required. Heuer designed a process to help analysts break this bad analytical habit, and we have built a prototype web application to help analysts apply Heuer's process in a co-located team setting.

The research, design, and implementation of ACH-W offered an opportunity to explore the following main themes:

- **Sensemaking**: general properties, cognitive challenges, and opportunities to develop support tools
- **Surface Application Design**: enabling co-located collaboration with large multi-touch surface displays
- **Agile Development Architecture**: support for rapid production and iteration of collaborative prototypes and apps

1.2 Organization of this thesis

The sensemaking literature describes efforts at various scales and in a variety of domains. In Chapter 2 we review some of the sensemaking literature, particularly as it relates to opportunities for tools and the cognitive advantage afforded by large surface displays.

Large multi-touch displays have been available for some time, but in recent years they have become larger, more powerful, and more affordable [33]. The collaborative benefits of these displays make them a natural fit for co-located teamwork. In Chapter 3 we describe our initial exploration of open source developer resources for surface application. We discuss design approaches as well as examples of Java, Python, and native development tools. We also begin to speculate on the featues of a web architecture might build on lessons derived from studying these older tools.

Chapter 4 opens with our first pass at a client-server architecture for supporting multi-touch. We then proceed to describe our adoption of Meteor [1] and D3 [8, 9], two new and powerful web development frameworks chosen their ability to assist in the rapid production of high-fidelity prototypes. We describe our use of data-driven visualization and discuss the benefit of the recent trend toward a component model for web design.

In Chapter 5 we describe our actual prototype of ACH-W, its motivation, requirements, design, and implementation, and then in Chapter 6 we discuss lessons learned from demonstrations and walkthroughs with professional analysts and we then apply some of their critique in a discussion of alternative designs.

We conclude with Chapter 7 where we outline the principles derived from this exploration and suggest areas for future research.

1.3 Contributions

The contributions of this thesis are:

1. A prototype collaborative sensemaking tool called ACH Walkthrough designed to leverage our research.
2. A set of interactive visualizations for ACH for use in a teamroom setting.
3. An exploration of web development architectures for building collaborative applications leveraging large surface multitouch displays.

1.4 Publications

Judith M. Brown, Jeff Wilson, Stevenson Gossage, Chris Hack, and Robert Biddle. *Surface Computing and Collaborative Analysis Work.* Synthesis Lectures on Human-Centered Informatics. Morgan & Claypool, 2013. My contribution was the authoring of chapter 6, which has been reprised and revised in this thesis in Chapter 3.

Judith M. Brown, Jeff Wilson, and Robert Biddle. A Study of an Intelligence Analysis Team and their Collaborative Artifacts. *School of Computer Science Technical Report TR-14-04 ,Carleton Uni.*, 2014. My work on this study was principally as an observer of two of four sessions, with some additional participation in the analysis. This

work helped motivate our choice of analytic technique, and it is therefore summarized in Chapter 5.

Jeff Wilson, Judith M. Brown, and Robert Biddle. ACH Walkthrough: A Distributed Multi-Device Tool for Collaborative Security Analysis. In *Proceedings of the 2014 ACM Workshop on Security Information Workers*, SIW '14, pages 9–16, New York, NY, USA, 2014. ACM. My contribution was as collaborating designer, and sole architect/developer. The UI design phase consisted of my own initial low-fidelity UI designs, followed by collaboration iteration to finalize the UI, followed by several rapid iterations of implementation and group review. The final result is presented in detail in Chapter 5.

Jeff Wilson, Judith M Brown, and Robert Biddle. Distributed Data and Displays via SVG and HTML5. In *Workshop on Distributed User Interfaces @ the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 67–70, 2013. My contribution was as principal author and sole software developer. Work for this paper appears in Chapter 4.

Jeff Wilson, Judith Brown, and Robert Biddle. Interactive Parallel Coordinates for Collaborative Information Analysis (In Submission). In *3rd Workshop on Programming for Mobile & Touch*. ACM, 2015. My contribution was as principal researcher, author, and sole software developer. This paper was prompted by reviews of ACH and is therefore reprised in Chapter 6 on evaluation.

Miran Mirza, Jeff Wilson, and Robert Biddle. Collaborative Annotations for Large Touchscreen Web Applications (In Submission). In *3rd Workshop on Programming for Mobile & Touch*. ACM, 2015. My contribution was in initial conception and early prototyping, as well as in later design contributions and reviews. This work appears under future ACH features, referenced in Chapter 6.

Peter Simonyi, Jeff Wilson, Judith Brown, and Robert Biddle. Supporting "What-If" in Touch-Screen Web Applications (In Submission). In *3rd Workshop on Programming for Mobile & Touch*. ACM, 2015. My contribution was in early project-specific

conception and prototyping, and then in consulation on project-independent plugin features. This work appears under future ACH features, referenced in Chapter 6.

Chapter 2

Background on Sensemaking

Sensemaking has been described as a process for transforming large amounts of data into new representational forms to respond to task-specific questions [65]. In domains ranging from the establishment technical training requirements to the production of national intelligence analyses, the activity of making sense out of vast quantities of raw data can best be described in terms of various theoretical perspectives.

This chapter examines three seminal papers containing foundational research applicable to the field of sensemaking and intelligence analysis. Our goal here is not to span the entire subject but rather to understand the sensemaking activity in general and to ensure that our own efforts using large multitouch displays are grounded in research, are well placed to support the analytical process, and are designed with attention to known challenges.

2.1 Introduction

The term *sensemaking* could hardly be more evocative. Physical sensory processes such as vision are excellent analogues for this deeply cognitive and social one. Faced with the proverbial hidden tiger, a physicist might describe our primitive ancestor's eyes as having been bombarded by a disrupted field of shady grey and orange light waves, but then what?

To produce meaning we might go on to describe the phenomenon of interpreting those light waves as the psycho-physical unfolding of a sign-producing process where the nervous system *encodes* raw light waves into progressively more meaningful pre-cognitive patterns. Early layers feed a series of neurologically encoded shape-detectors, and then as the resulting patterns emerge they proceed to activate notions of texture, then structure, then animal, then big animal, then big cat, then (hopefully) reason to retreat.

Similarly, research into sensemaking has identified layers of processes (at the level of the individual and the group) that encode and categorize raw data into increasingly meaningful forms of information used in subsequent reasoning processes. These overall processes can

support a wide range of activities, including anything from the authoring of consumer product recommendations or the briefing of world leaders weighing their response to rapidly unfolding international events.

So the connection between *sense*-making and *sensation* should be clear. Perhaps less obvious is the importance of the second part of the term: *making*. In the literature on the subject we find support for the idea that sensemaking is an activity comprised of *reactive* and *contructive* transformation of raw data that may offer as much in the process as it does in its end products. Indeed the very papers selected for this review show evidence of this process in their own attempts to make sense of their subject matter.

We first look at a paper that explicitly address sensemaking [65]. It explores the cost tradoffs in the task of transforming raw data into useful intelligence. The next paper that elaborates on the sensemaking process in the domain of intelligence analysis [61]. And the final paper explores the value of large displays in merely providing space as a substrate for thinking [4].

These papers relate to, either directly or indirectly, broader research themes in human-computer interaction. These include distributed cognition, activity theory, embodied cognition, mental models, metaphor, semiotics, and more. The discussion section will attempt to identify those frameworks that provide the best guidance in this space.

2.2 The Cost Structure of Sensemaking

Russell, D. M., Stefik, M. J., Pirolli, P., and Card, S. K. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human Factors in Computing Systems*, ACM (1993), 269–276.

This first paper introduces the activity that has come to be known as sensemaking. The 1993 paper by Russell et al. described a general form of the sensemaking process, namely the *learning loop complex* (Figure 2.1). The authors sought to characterize and quantify the cost tradeoffs in sensemaking, particularly in the context of large amounts of data.

Central to the learning loop is the iterative design of a representational schema used to encode raw data into a form that can be used for later reasoning.

The paper describes sensemaking in a wide array of contexts, including such fields as course development for technical training in one case and high school algebra in another,

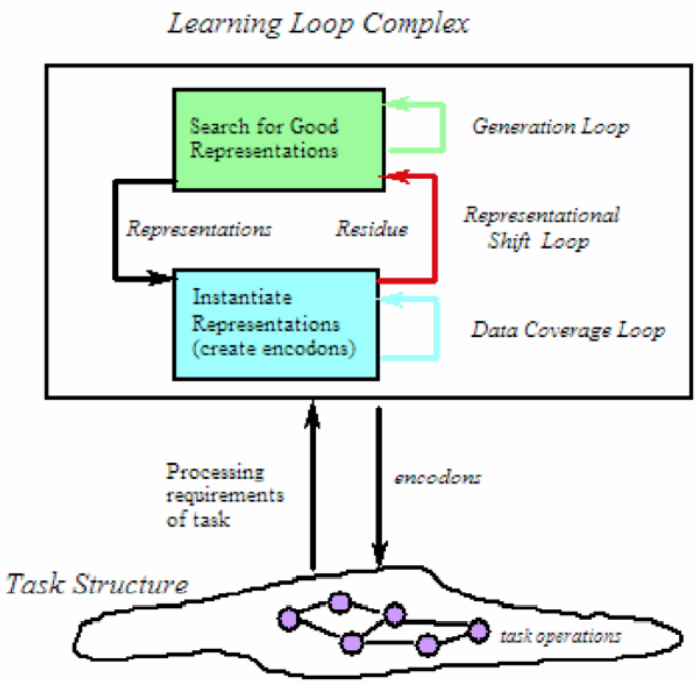


Figure 2.1: Modeling Transformations in the Learning Loop [65]

evaluation of the research potential of a technology, and business intelligence analysis for the production of periodicals. Their material was drawn from internal technical reports from Xerox PARC and also from external field studies.

Their central case study demonstrates the potential for value in performing such detailed cost modelling. The study describes an undertaking by a team of analysts to overhaul the training courses for printer and scanner technicians at Xerox. The content of their new courses had to be extracted from a mountain of documents, none of which were explicitly designed to support training. Several months of data collection and processing would be required, and the case study shows remarkable forethought, preparation, and flexibility in their efforts to minimize data collection costs and maximize course quality.

The lead group in the case study quickly recognized the need for an updated conceptual model of relevant technologies, and they took two months to iteratively design the schemas that would be used in subsequent extraction and encoding.

The second phase of the effort consisted of five months spent by three teams on data extraction, where the raw documentation was collected and encoded into databases designed by the lead group. The final phase was comparatively brief, but represented an immense

contribution to the overall activity. The course planning team employed computer tools to iteratively identify and organize the central course topics.

It is worth noting two essential features of this exercise. First we note that the provisional schemas provided by the lead team were updated during the extraction phase as newly discovered data put pressure on their models. In Figure 2.1 this is the notion of *residue*, those elements that don't fit the provisional representation and therefore trigger a representational shift. This feedback loop and the resulting evolution of understanding are what give the process its adaptive power. Secondly we point out the fact that this exercise was fundamentally collaborative in nature, a feature that drew as much from its need for a variety of skills as it did from sheer scale.

The effort spent on producing and maintaining a high quality representation in this case study enabled the subsequent use of computer support tools to achieve their final results. This constitutes one central theme of the paper: the costs incurred in transforming raw data can pay off immensely in computation, visualization, and interaction with task-specific representations. We note that there is a certain element of risk in conducting several months of encoding when the payoff is uncertain, but some of this risk could be mitigated with the use of pilot studies and off-the-shelf visualization tools.

The authors studied more than this one case in detail. In fact they compared it with several other sensemaking activities and found enough important similarities to confirm that their *learning loop complex* adequately described the activity of making sense of information. Their paper show side-by-side comparisons of several case studies and the pattern is indeed apparent.

They then delved into cost structure in one-off cases such as the course development example and recurring cases such as periodic generation of financial intelligence reports.

In discussing the cost analysis for one-off activities they raise another theme in the paper, namely their characterization of sensemaking as being based on an *anytime principle*. The authors borrow a term from computer science, where *anytime algorithms* are computations that are designed to provide the best solution at any point in time. The more time (or computing resources) they are given, the better the solution they provide. An appropriate analogy might be found in playing the game of chess, where the difficulty of one's opponent (automated or otherwise) greatly depends on the amount of time they are given to think.

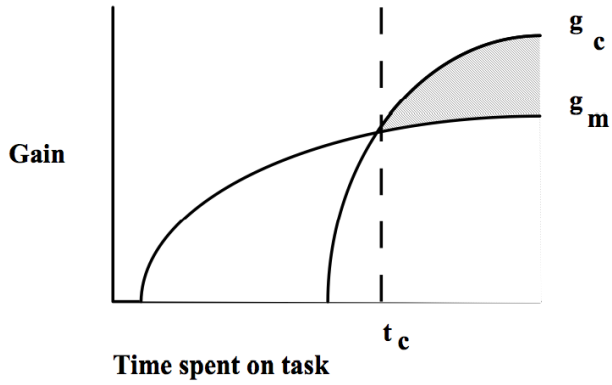


Figure 2.2: Manual (g_m) and computer (g_c) gain tradeoff [61]

In their printer technician training example, this anytime principle applied to the computer support used in cluster analysis. Where it might have been possible to reason through the topics manually, producing at any time along the way a passable approximation of the ultimate topic structure used, the use of computers greatly increased the number of iterations the researchers could use to re-group the topics and visualize the results. In the course development case study, this led to greatly improved topic structures.

Note that the concept of "gain" (e.g. in improved course structure) is difficult to precisely quantify, and it certainly differs from one sensemaking challenge to the next. Nevertheless the transformation of raw data into task-specific representations must take place for reasoning to occur, and the authors provide simplified model to illustrate when computer-based methods of doing so provide value over manual ones (Figure 2.2).

In simple terms, the computer based strategy should be used only when the deadline t_c comes after the crossover where the gain from manual methods is exceeded by the gains from computer-based strategies. Clearly there is value in estimating the crossover point *before* undertaking the encoding and calculating activities required for the computer-based solution, but an awareness of the computability of certain tasks based on the known inputs and the known quantity of raw data should help guide the decision. For example in the printer case, the synonym matching and the distance weighted topic clustering that were called for by the nature of the sensemaking task were computational challenges that could be both: 1) reasonably anticipated to have value, and 2) easily deemed impractical if undertaken manually.

The mathematic model the authors used to describe recurring tasks was drawn from an unexpected source. Biologists examining the tradeoffs between different survival strategies of animals in the wild have developed mathematical models described in *foraging theory*. Recognizing that people must also make choices between possible subtasks in sensemaking, Russell et al. apply similar mathematical models to quantify tradeoffs in the information foraging behaviour of human researchers.

The contributions of this paper as reported by the author are the identification of the *learning loop complex*, the characterization of sensemaking as an *anytime* activity, and the central feature of *representation design*.

The *learning loop complex* is a powerful model that informs this entire literature review. The notion of *representational shift* has both short and long term implications in the understanding of complex domains. The concept of feedback pressures from the uncharacteristic data in the form of *residue* and from the task in the form of *processing requirements* (not to mention schedule pressure) are the central insight that we find repeated in one form or another in all these papers.

This particular paper also shows the value of flexible tools that might support this essentially *constructive* effort. There are several references to the value of a "representational shift". For example they describe a direct manipulation interface that helped course designers organize topics by prerequisite, noting that this computer support would not have been available if they had not "shifted" into an electronic form of encoding their subject matter.

Interesting themes from this paper include: 1) the establishment and maintenance of *mental models*, 2) the use of *metaphor* in assisting with representation (their own use of biology research on foraging behaviour as a model for repeated searches is an excellent example), 3) the aspects of *distributed cognition* in the teamwork described and in the *cognitive offloading* of computation made possible by good representational schemas, and 4) the value of *visualization* and *interaction* in finding new insight.

2.3 The Sensemaking Process

Pirolli, P., and Card, S. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International*

Conference on Intelligence Analysis, vol. 5 (2005), 2–4.

This paper comes from Pirolli and Card (2005) [61], who have created a sensemaking model using a cognitive task analysis (CTA) of intelligence analysis interview data. Using a think-aloud protocol, the authors set out to understand the analytic process in greater depth.

Their first observation recognized the role of *schemas* in making sense of large volumes of information. They drew connections to recognition-primed decision making, a process that improves with greater experience. Among other sources, they cite research from Simon and Chase [67] in the improved memory ability of chess masters for boards that followed meaningful patterns. Through a combination of experience and categorization, experts become adept at *selecting* particular aspects of the raw information and *encoding* it in task and domain specific representations.

They noted that analysts often produced and maintained cognitive aids (collections, maps, lists, etc.) that served to amplify their ability to find patterns and organize information as it relates to their particular domain. These *expertise schemas*–as Pirolli and Card called them–were not fixed, but rather were dynamically adjusted under the demands of new data and new analytic tasks.

We have seen this feedback effect between raw data and the schemas used for analysis characterized by Russell et al. [65] as the *"learning loop complex"*. We briefly repeat that the process of characterizing or *encoding* new data gives rise to a *residue* in the form of items that don't fit the early provisional schema. As the analytic process proceeds, analysis of this residue assists in modification of the schema.

Through their CTA, Pirolli and Card elaborated on the process, clarifying more intermediate storage structures and characterizing the goals associated with the various transformations. The result was a *notional model* that distinguishes two inner loops of activities: 1) a foraging loop (Figure 2.3, left half), and 2) a sensemaking loop (Figure 2.3, right half), and one larger reality/policy loop that encompasses the entire process (Figure 2.3, full).

The foraging loop is concerned with seeking, filtering, and extracting activities, and its overall aim is the identification and possible schematization of *evidence*. The sensemaking loop involves the iterative production of a *mental model* that best fits the evidence.

The larger reality/policy loop is presented in the top-down and bottom-up activities as depicted by the bidirectional cascading arrows in Figure 2.3. The initial bottom-up process

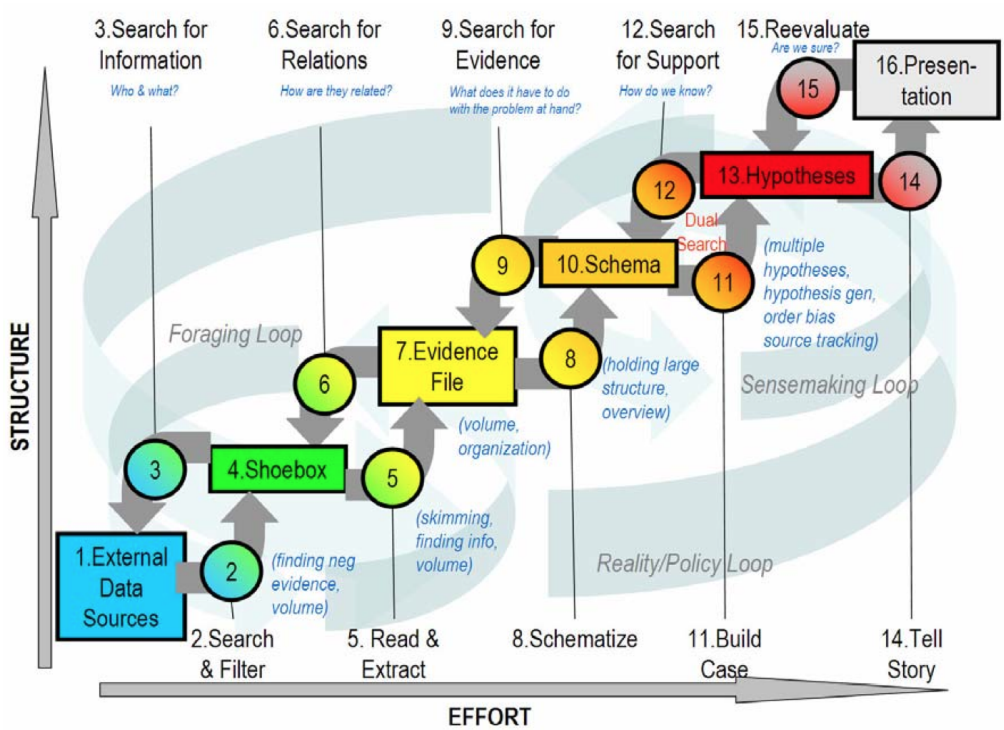


Figure 2.3: Notional Model derived from CTA [61]

is shown left to right along the bottom of the figure: *search and filter*, *read and extract*, *schematize*, *build case*, and *tell story*.

The top-down process (right to left along the top) tends to proceed after evaluation from peers or clients, and includes the following activities: *re-evaluate*, *search for support*, *search for evidence*, *search for relations*, and *search for information*. These forces represent the need for iteration and improvement, and are essential to the improvement of a given representation over the course of a project.

It is important to note that the data structures shown in the model (the boxes in Figure 2.3) can be formal or informal artefacts, and in some cases they are only provisionally created if they are created at all (as we will see in the *shoebox* and *evidence list* examples in the next paper in this review). Regardless of their actual presence as artefacts, they do exist in notional or cognitive terms and serve as useful structural elements for describing the process.

The second half of Pirolli and Card's paper explores the points of leverage where technology might make a positive contribution, particularly in cases where analysts are confronted

with massive volumes of data. They identify the following leverage points:

Exploration-enrichment tradeoff The risk of missing relevant details imposes a cost of more inclusive search parameters. The solutions they recommend use focus+context techniques such as Fisheye [26] lenses, where the design allows users to focus on a particular part of the display while also seeing the broader context.

Scanning, assessing, selecting Application of pre-attentive highlights can greatly improve this expensive process. Note that this requires some form of pre-existing schemata but we might assume that this could be achieved without undue complication or distraction.

Attentional shifts The authors discuss the heavy costs of shifting attention that come from task re-assignment, interruption, discovered anomalies, etc.

Follow-up searches Previous searches could be stored, making subsequent searches easier to achieve. Note that this can be previous result sets or unpopulated queries with predefined relationships.

Span of attention Solutions to limited span of attention include cognitive offloading and information visualization. (The next paper in this review also solves this by providing analysts with screen space to think.)

Generation of alternative hypotheses The authors indicated a need for improved tools for generating, managing, and evaluating hypotheses.

Cognitive bias They call for new tools that distribute analyst attention toward *highly diagnostic* evidence and to search for *disconfirmation* rather than confirmation.

Pirolli and Card's highly cited research provides a *notional model* of analyst sensemaking that can be used for a variety of research purposes. The model characterizes and names the phases, goals, by-products, and feedback loops within the overall analytic process. They also distinguish a sensemaking loop from its related foraging loop. The model provides a useful framework for situating, designing, and evaluating analytic support tools and processes.

The notional model stands on its own as an important contribution in evaluating processes and tools that support sensemaking in intelligence.

The authors also identify a set of "pain points" where technology might provide the greatest leverage. These points make use of the common themes that emerge throughout the model, but they also demonstrate an awareness of the particulars of this challenging environment. Citing such issues as the risk of *cognitive bias*, the cost of *attentional shift*, and the challenge of generating *alternative hypotheses*, [1] they simultaneously delve into and rise above the particulars of the steps they have identified.

2.4 Space to Think

Andrews, A., Endert, A., and North, C. Space to think: large high-resolution displays for sensemaking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 55–64. ACM, 2010.

In this final paper from Andrews et al.(2010) [4] we look at the contribution to sensemaking of large display surfaces with high resolution. The large display environment they created was a 4×2 array of 30-inch high resolution displays (Figure 2.4). The context of this research was explicitly related to the Pirolli and Card paper above. In other words it sought to characterize the value of their display technology in the service of some of the "pain points" identified in the sensemaking task in the domain of intelligence analysis.

The authors conducted two studies. The first was a between-subjects quasi-experimental design with eight participants that compared the large display condition with a standard 17" single display condition. All participants were also given access to scrap paper and a whiteboard. The analysis sessions were uninterrupted by "think-aloud" protocols, and instead semi-structured interviews were conducted after completion of the sensemaking session.

Participants performed analytic tasks using data from a VAST 2006 conference contest. The value of this approach was that it was designed to be brief enough for a single session and easy to score for correctness. The scores at the end of the study did not reveal any significant effect of display size, but there were a number of reasons aside from mere statistics why this would be the case.

[1] The process of Analysis of Competing Hypotheses, the subject of our prototype, sets out as its raison-d'etre the issues of cognitive bias and the need for alternative explanations.

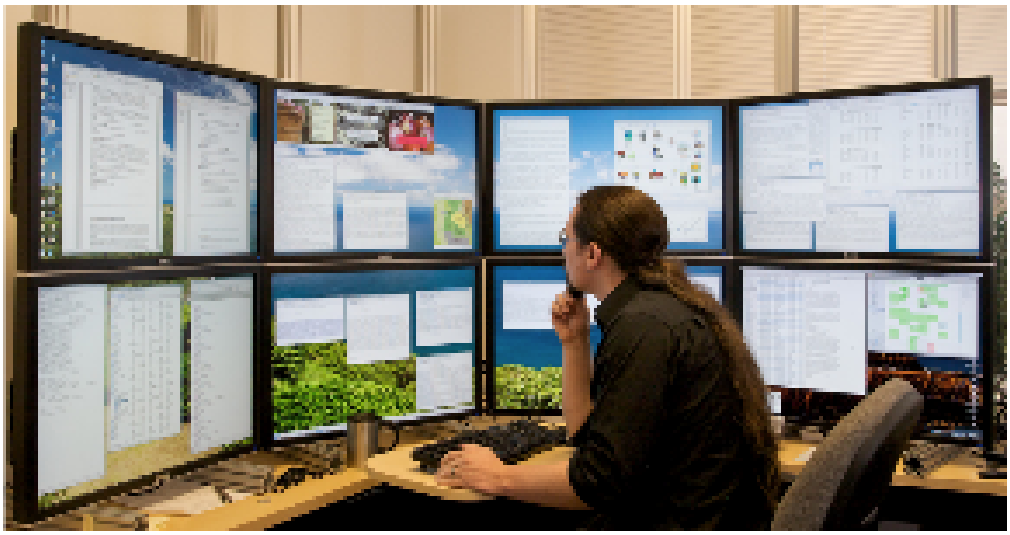


Figure 2.4: Analyst workstation (10240x3200)

Of much greater value in this first study were the qualitative results describing participant behaviours with respect to interacting with their environment. These were primarily related to management of documents opened in WordPad, where the small-display condition relied on the Windows taskbar to switch between documents whereas the large display group made use of the available space to organize and display these artifacts.

Also interesting was the creation of artifacts by the two groups. In the small display condition, participants produced electronic documents resembling a *shoebox* but they also made notes on paper that collectively represented an *evidence list*[2]. In the large display condition there was an almost complete *lack* of paper notes, and the electronic artifacts they produced resembled the evidence lists that had required paper in the small-display group, while the notion of a *shoebox* could be found in the structural arrangements on the display.

The second study in this paper delved more deeply into characterizing the use of space for sensemaking. It involved observation of four practicing analysts and one developer of analytic tools, all of whom used the large display. Their use of the workstation was analyzed along the following themes:

External memory Several behaviours were observed that confirmed the role of display size in the maintenance of external memory. These included "glancing while reading", where

[2]See Figure 2.3 for the context of the *shoebox* and *evidence list/file* artifacts.

a participant would glance over to another part of the display to check details. Another was re-reading and scanning that occurred during pauses, suggesting a process of assimilation and schematizing (recall the *learning loop complex*). In one particular case the authors describe a serendipitous discovery during a scanning session of a person's name that was a key figure in the intelligence scenario, something that would not have occurred without the availability of several open documents.

Atomization of information Rather than rely on the production of new artifacts (the electronic or paper evidence files from the first study), these partipants tended to use highlighting techniques (e.g. with colour or bold fonts) to form a loose collection of relevant information.

Organizational strategies Documents tended to be laid out in evolving semi-structured patterns, with a central workspace surrounded by a variety of documents organized using a variety of spatial strategies the arrangement of which supported the categorization and marshalling of evidence. The space was used in a variety of ways representing such concepts as timelines and criticality. In one case the participant started with a timeline in the form of open documents along the x-axis of the display without losing this arrangement she shifted documents vertically to support classification by theme.

Integration of Process and Representation As Pirolli and Card described in their paper, analysis tends to be iterative and free-flowing between levels or stages of analysis. Andrews et al. identified several factors in large displays that supported the fluidity of these shifts in focus. One particular example of this was in the arrangement of these highlights within documents. Not only did this provide the details of potential evidence, but the distinct patterns formed by the highlights had the effect of making documents easier to identify on the desktop. Notice how this resembles the interaction technique of abstract/elaborate in the visualization paper, and helped analysts use effortless perception rather than effortful searches.

The paper offers two contributions to this literature review: first its observational study demonstrates that simply by changing the human factors in a physical environment it is

possible to observe interesting cognitive and behavioural effects even in such a complex environment as intelligence analysis. Second, the nature of the observations offer excellent illustrations and explanations of the notional model provided by Pirolli and Card.

Of particular interest for this review was the effect of space on the production of artifacts, and the "accidental visualization" formed by highlighting sections of documents. The identification of a central work area surrounded by open space for flexible ad-hoc reasoning will likely inform future designs.

2.5 Summary

Between these three papers we find guidance for a number of useful observations that may feed into design principles.

From Russell et al. we learned the value of the *anytime strategy*, where one can expect to be called on to produce a result at any point in the process. This idea depends on an iterative process of refinement, one that improves with time. We also saw our first description of user-driven encoding, an important hint that flexible categorization schemes are required.

From Pirolli and Card we saw a refinement of Russell et al. for intelligence analysis, and the recognition of iterative phases of collection and analysis. We learned of a number of "leverage points" where technology could be applied to good effect, and we saw confirmation of Heuer's concerns about *cognitive bias*.

From Andrews et al. we learned that the simple expediency of offering digital space had important cognitive benefits, making it possible to reduce the need for external artifacts in a digital environment. We also began to understand the provisional nature of the sensemaking structures Pirolli and Card called the *shoebox*, the *evidence file*, and the *schema*, and how their malleability may be an important property.

Chapter 3

Background on Surface Application Development

In domains where the technology infrastructure is rapidly changing, interaction design and software development are closely linked activities. In this chapter we first briefly review a few good approaches to the development process that address issues pertinent to surface applications for analysis work. We then discuss user interaction toolkits for development of surface applications, with an emphasis on open frameworks for large surfaces important for collaborative analysis work.

3.1 Application development processes

As we will see in Section 3.2, surface applications demand a paradigm shift with respect to interaction design. Clearly the topic of application development processes can fill an entire bookshelf, and so we cannot hope to address the whole domain. Instead we briefly touch on a few ways to incorporate interaction design and empirical evaluation into the development cycle, in particular where they apply to enabling visual analysis, collaboration, and rapidly evolving requirements.

3.1.1 Designing for visual analysis

Data analysis is a domain where analysts have significant expertise, and where application domains often have well-established conventions. While there is much general knowledge about information visualization, it is nevertheless important for any software development effort to work closely with analysts in the relevant domain. For example, Conti [17] presents a development process in chapter 10 of his book *Security Data Visualization* that specifically addresses software development for visual security analysis. The process combines the expertise of visualization experts, user experience specialists, security experts and software developers. It considers hardware issues, memory issues, usability issues, software architecture issues, and domain issues, and helps experts from multiple domains coordinate their

work.

At the heart of Conti's work is the awareness that, "By definition, humans are integral to the visualization system." In stating this he admonishes teams to never lose sight of the human while solving computer challenges. This leads to an iterative user-centered design process:

1. **Focus on Users/Tasks**

   Understanding users' goals and the tasks they must perform to achieve them is an essential element of user-centered design. The first challenge in visual analysis is to capture tasks that users may have difficulty articulating, particularly when there are aspects of discovery in the task. The next challenge is to keep users engaged in the minutiae of a sometimes laborious design process. This can be difficult with expert users whose time may seem best spent on their real work. This can sometimes mean identifying or training appropriate proxy users who can take the place of actual domain experts for certain tasks.

2. **Perform Empirical Evaluation**

   Designs should be validated early, involving domain experts (or their proxies) and interaction design professionals. Use prototypes extensively, starting with low fidelity paper-based ones, and gather data to validate design assumptions. When moving beyond paper-based prototypes consider using development tools that will make it possible to reach a broad range of usability testers.

3. **Iterate**

   Solutions should be built incrementally, incorporating lessons learned from earlier analysis and evaluation phases.

3.1.2 Designing for rapidly evolving requirements

Changing requirements will likely be a hallmark of surface application design and development, because there are so many unknowns. End users will only know what they want in an application as they begin to comprehend the possibilities. In the third edition of *Interaction Design* [63], Rogers et al. claim that *agile software development* [16] practices offer

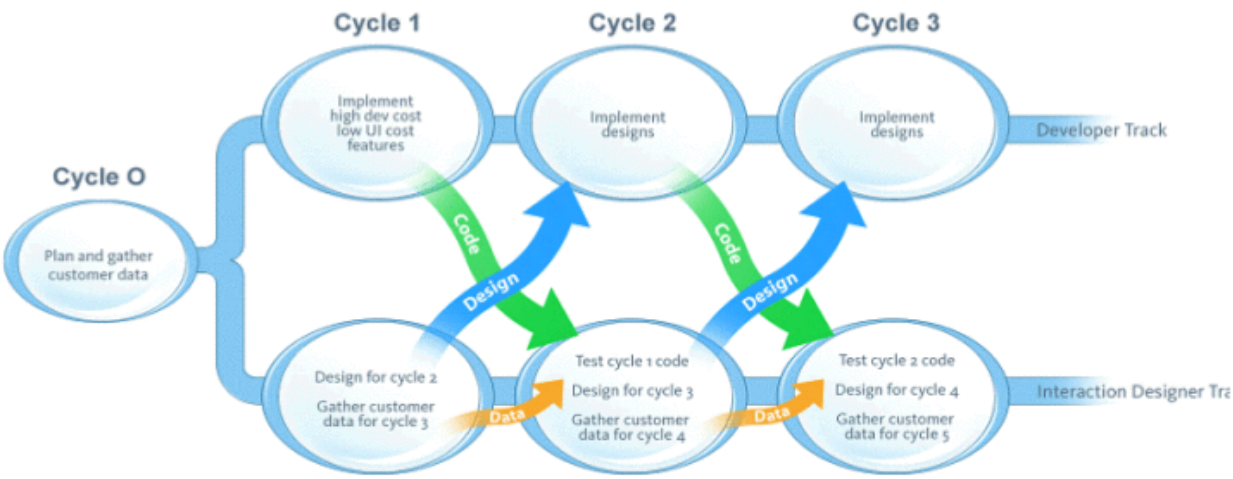


Figure 3.1: Interleaved parallel tracks in interaction design and development work. Source: Journal of Usability Studies [70]

the most promising strategies to integrate design into the development process. They acknowledge the variety of distinct agile methodologies, but point out that these strategies share similar benefits in their use of early and regular user feedback, their ability to handle emergent requirements, and their ability to balance flexibility and structure in a project plan. In brief, the main phases of the interaction design process they describe are: 1. Establish Requirements, 2. Design Alternatives, 3. Prototype, and 4. Evaluate. Because design and development work for surfaces is so novel, and therefore more unpredictable, agile methods appear very suitable.

Rogers et al. also raise the issue of when to begin work on implementation activities, particularly when combining the efforts of designers, developers, and users/evaluators. The underlying issue is that the interaction design and the software development are *both* iterative processes. They suggest one way to maintain productivity among these groups is to interleave their activities in parallel tracks (Figure 3.1) in a manner described at Toronto-based Alias Systems (now owned by Autodesk) [70]. They call their iterations "cycles". In Cycle 0 the most important features are investigated and prioritized. In Cycle 1 the developers work on features requiring little design input while interaction designers produce their initial designs and gather user data. In subsequent cycles the developers implement features designed in Cycle n-1 while interaction designers test code implemented in Cycle n-1, design features for Cycle n+1, and gather customer input for Cycle n+2. By interleaving the activities of

designers, developers, and customers all three groups are able to maintain a high level of communications and productivity.

Agile methodologies gain several advantages from their user-centred focus, but perhaps the most powerful is the frequency of evaluation. It should be noted, however, that while user input is essential, users typically lack the formal background to fully evaluate prospective design solutions. Therefore, while it is essential to include users (or suitable proxies) in the establishment of requirements it is also important to apply due diligence with empirical and heuristic techniques. Examples of such techniques include: Wizard-of-Oz testing with low-fidelity prototypes that uncover issues as early as possible; cognitive walkthroughs that guide analysts through potentially challenging tasks with a focus on mental models; and heuristic evaluations that examine the overall product with respect to established dimensions of usability. Neilsen and Mack refer to these as *inspection techniques* [56]. Nielsen also refers to these as *discount* evaluation techniques, naming them for their low cost and their good rate of problem discovery, both features that increase the likelihood that they will be used in spite of schedule and budget constraints [55]. In novel design situations frequent evaluations are particularly important.

3.1.3 Usability analysis techniques

Designing software for collaboration introduces an important new dimension that can complicate evaluation efforts. Pinelle, Gutwin and Greenberg [60] point out that traditional discount evaluation techniques (e.g. inspections, walkthroughs, and heuristic evaluations) successfully address *individual work*, however they fail to model usability aspects specific to *teamwork*. To this end, they offer a framework called *Collaboration Usability Analysis* (CUA) to address this essential distinction.

Pinelle et al. use standard observational and interview techniques in a real-world work setting to discover what they call the *mechanics of collaboration*. The *mechanics* that are captured include communication tasks such as *explicit communication* (e.g. speaking, pointing, drawing, etc.) and *information gathering* (e.g. basic awareness of others, changes to objects, eye gaze direction), and coordination tasks such as *shared access* (e.g. taking tools, occupying space, reserving resources), and *transfer* (e.g. handing-off or depositing work items).

Having an explicit language to capture these mechanics allows designers to model these

in diagrammatic forms, taking care to distinguish *taskwork* (i.e., private or individual tasks) from *teamwork* (i.e., group-related tasks). They offer a diagram model of task flows with Boolean AND, OR, XOR paths, and they also capture branching and iteration. The goal is not to produce a fixed structure representing a proscribed group dynamic, but rather to capture the essence of each task and represent its value to group objectives. The model can then be used as a resource for evaluating proposed software designs using discount methods. The model may be able to clearly show that the new design is better in some specific way, such as by providing better shared access to resources, in comparison to the old design.

In certain cases there is value in very detailed analyses of user behaviour. Examples include analyzing data mined from system logs, or capturing the detailed mechanisms of collaboration through detailed observations as mentioned above. An alternative approach is to use eye-tracking. The principle is that, often, where we are looking is where we are attending, and where we are attending is especially important in collaborative activities.

For conventional desktop computer systems, Human-Computer Interaction specialists have used eye-tracking to study what users are looking at while using the software. This helps researchers link gaze with the current state of a user's mind, and can provide insights into such things as distractions. This can be a useful technique during a usability study to determine how to improve layout and information architecture [49]. During usability tests a researcher could use eye-tracking to examine the amount of fixations or duration of fixations to discern objects of interest on a display. Alternately, Komogortsev et al. suggest that excessive eye scanning could imply a usability problem [44].

For collaborative systems, such as those with large surfaces or multiple devices, eye-tracking can also be useful. When individuals are co-located and communicating, both eye expressions and eye directions are components of interaction. For example, the direction of a pupil's gaze is often leveraged by teachers to ensure that students are attending to the proper material during a lesson. Head-mounted eye-trackers allow tracking of gaze around a large screen, between smaller devices, and beyond to the general environment.

Studies of eye-gaze to study social interactions should always be contextualized. Humans are naturally a social species capable of highly complex social structures and procedures. Communication itself involves more than just verbal or written language; it involves non-verbal cues, contextual cues, environmental cues, and even time-based cues. Although highly

complex, these behaviours are not random, as in order for communication to have structure there must be a solid common understanding for it to occur. Our understanding of communication is influenced by Clark and Brennan's concept of *common ground* [14]. In this framework the constant back and forth flow of information during conversation is referred to as utterances, a person's contribution to the conversation (typically verbally, though non-verbal contributions are also relevant). Eye movements and expressions are a part of this rich social interchange.

3.2 Application development frameworks

The early 1980s brought the revolutionary Xerox Star interface [69]. Since then, personal computers and their operating systems have co-evolved to support the desktop GUI paradigm known as WIMP, an acronym for Windows, Icons, Menus and Pointers. While all these terms are pluralized by convention, the fact is that desktop systems support only one cursor pointer. In spite of multi-touch trackpads' appearance on Apple's MacBook Air in 2008, or PCs that support a mouse, a trackpad, and possibly even an isometric laptop joystick, all these controls actually refer to a singular cursor.

With the advent of the collaborative tabletop [54], the latest successor to the desktop and the laptop, we can now take the plural form "Pointers" literally. This trivial grammatical observation represents a significant engineering challenge. The management of multiple simultaneous users around a single display has implications all the way from hardware detection to window design. In some ways it is the computer equivalent of installing multiple steering wheels on a car. Extending the automotive analogy a little further, operating systems are gradually improving their support for multiple simultaneous controls, but these are typically still in service of a single human driver. Support for multi-touch gestures is getting stronger, but the ability to distinguish one person's input from another's is typically outside the scope of today's operating system requirements. A number of system architects have understood the features essential to surface collaboration and new development tools and frameworks are emerging every year. The WIMP approach emerged about the same time as Object-Oriented Programming (OOP), and despite framework design diversity that continues to this day, there was symbiotic relationship between WIMP and OOP that facilitated some consensus. In surface computing there is little sign so far of anything similar.

3.2.1 Features of multi-touch development frameworks

Enabling surface interaction requires more effort than simply running applications designed for the mouse on hardware capable of detecting touches. Such applications do not take advantage of multi-touch capabilities, are not sensitive to gestures and do not support collaboration well.

We have considered a number of multi-touch frameworks for our prototype work on analysis tools. We have had a preference for cross-platform support because it enables our team members and users to work with platforms they are already using. In practice, cross-platform support has also meant open-source frameworks because there are no cross-platform proprietary frameworks. An advantage of open source is that multi-touch is still evolving, and wider input helps to be able to advance the state of the art. Frameworks that we used for any duration were chosen for their focus on surface collaboration, and for the maturity of their framework. We have come to see the following features as essential to the creation of multi-touch multi-user surface applications.

1. **Tabletop Canvas**

   Multi-touch tabletop interfaces are based on a drawing metaphor typically known as a canvas. The canvas takes the place of an operating system desktop – the standard WIMP design – and introduces support for rotation and resizing along with new multi-cursor modes of input. The canvas is automatically repainted many times per second, generating the perception of an animated surface.

2. **TUIO input provider**

   For applications that require touch events from devices across a port (or if touch events need to be passed across the network) there is a transport layer communication protocol for touch and motion data called TUIO [42]. Riding on a network protocol called Open Sound Control [15] originally designed for transmitting events from musical synthesizers and other multimedia devices, the TUIO protocol has become the de-facto standard for sharing multi-touch cursor events from a number of different event sources.

   One important advantage of using a network protocol to send touch information is the ability to forward touches to another machine. This helps enable very large displays to be built from arrays of smaller ones (Figure 3.2).

Figure 3.2: Large multi-touch, multi-person displays. Source: http://multitouch.fi

3. **Ability to define and incorporate new input providers**

   The provision for new local input sources enables new modes of interaction. These sources can include local inputs such as Microsoft Kinect and Leap Motion, or remote inputs from real-time networked collaborators. Input might also require proprietary drivers for receiving gestural input from camera images or from accelerometers. Sometimes interaction events will be transmitted through the operating system or indirectly through browser applications. In general it must be possible to define and configure the varied sources of touch events to applications.

4. **Touch optimized UI elements with support for rotation and resizing**

   User interface widgets must support rotating and resizing so they can be used comfortably on all sides of a tabletop system and with a reasonably large number of collaborators. Ideally it should be possible to change their visual style based on tags or descriptors such as we find in HTML with CSS. The inclusion of layout managers for grouped UI elements is also helpful. Widgets must also support high resolution text elements.

5. **Gesture definition, localization and prioritization**

   Definition and recognition of custom multi-touch gestures presents several challenges. In essence, sets of otherwise independent events must be interpreted as joint events. Gesture points must be grouped when they are within a reasonably small range of each other, allowing multiple simultaneous users to perform multi-touch gestures independently. In the event that a given touch is ambiguous, for instance if it falls inside the

range of more than one gesture, then the system must be able to prioritize its assignment to a specific gesture according to an appropriate set of rules. This calls for the definition of ad-hoc "cursor collections" that map each cursor to exactly one gesture.

6. **Graphics and multimedia**

   Multi-touch frameworks can be measured by the extent to which their API reduces the complexity of encoding interactive graphics. The necessary simplification of an API can sometimes limit outlying cases, and so some access to the underlying hardware accelerated graphics library (i.e., OpenGL) can also be useful. Multimedia hooks are also useful for creating mashups with rich streaming content.

7. **Flexible Event-Based Architecture**

   Events in multi-touch applications can have much more nuance than the pointer-based idiom of traditional GUI frameworks. It is important to be able to react to events raised by partially completed gestures, off-target or out-of-bounds gestures, gesture timer expirations, and so on.

The next set of features may not be essential but they are certainly nice to have.

1. **Layout Language**

   An extensible language for application definition can greatly improve maintainability.

2. **Multiple virtual keyboards**

   Multiple users should be able to enter text independently into their own separate windows.

3. **Multiple cursor pools**

   It should be possible to group cursors according to their source. For instance it should be possible to distinguish between remote cursors and local ones, and it should be possible to distinguish between pen input and mice or touch points. These distinctions are not necessary for simple operations, but they are quite useful for providing gesture processing with important collaborative dimensions such as a participant's identity.

4. **Plugin gesture processing**

   Gesture interpretation can be achieved a number of ways. Most frameworks will have some sort of predefined gesture engine, but ideally it should be possible to use custom engines that use techniques such as rule-based processing or machine learning.

5. **Physics Library**

   Simulated physics adds a measure of realism to multi-touch interfaces. Some elements of this approach were explored early, such as the inertial scrolling seen in the experimental Star7 personal digital assistant demonstrated in 1992. This has now become a commonplace feature, especially in smartphones and tablets. The idea is to provide an experience that better matches the experience of an actual control wheel for scrolling documents and lists. The more general appeal of simulated physics has been shown in experiments for drag and drop on the desktop [3], and being actively explored for multi-touch interfaces.

3.2.2 OS-specific frameworks

Some platforms represent end-to-end designs that include native tools. Examples of this include SMART Technologies' SMART API, MultiTouch Ltd.'s Cornerstone SDK, Google's Android SDK, Apple's UIKit framework, and Microsoft's Surface 2.0 SDK. When working with such hardware the advantage of commercial tools is often worth the expense. Software features are well-matched with hardware capabilities, and documentation and support are often available.

Microsoft created a broad set of tools for their Surface 2.0 hardware, and the same tools applied when the "Surface" brand was re-assigned from their large multi-touch table to their tablet. The Surface 2.0 software development kit (SDK) extends the Windows Presentation Foundation (WPF) to provide the essentials of redesigned UI elements, multi-user support, etc. that we identified in Section 3.2.1. For example, the producers of the Surface SDK understood the need to redesign their interface elements (buttons, sliders, text boxes, containers, etc.) given the new challenges of surface interaction.

In some frameworks with native OS integration, such as the Qt Framework [57], we find a more gradual transition from a strictly WIMP paradigm to a mix of WIMP and multi-touch

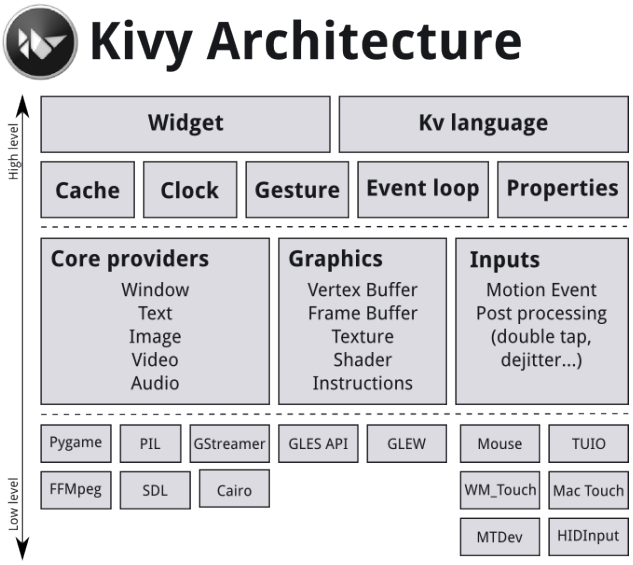


Figure 3.3: A platform for rapid cross-platform multi-touch applications. The Kivy Architecture. Source: kivy.org

support. In modifying existing interface elements the result is a mix of interaction paradigms that presents implementation challenges for developers, who must carefully choose from a growing set of widget features to produce an appropriate effect in the appropriate context. By contrast, Nuiteq's Snowflake Suite [58] is an example of a commercial framework designed from the ground up with multi-touch in mind.

3.2.3 Python-based frameworks

In our work we have focused on development environments that promise cross-platform support and rapid prototyping of multi-touch applications. The Python language Kivy library and its predecessor PyMT are examples of frameworks designed for rapid development of multi-touch applications. They support all our essential requirements and some of the others we considered desirable. Both projects are open-source.

Kivy has an active developer community with a demonstrated commitment to cross-platform development with initial input support for Windows, Mac OSX and Linux, and more recently with additional support for Android and iOS. Our experience with its predecessor PyMT (under Windows and Linux) confirmed our hope that it was possible to rapidly design prototypes with good performance and a polished look and feel. The Kivy graphics support engines provide a simplified API for newcomers to graphics programming. The expandable

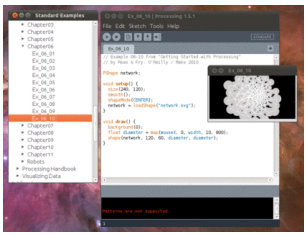
design made it possible to plug in new input sources and libraries for rendering charts and graphs.

PyMT came with a number of core UI widgets that could be easily rotated and resized for use on tabletop systems. They also provided a simplified class that could inherit these abilities, enabling the design of customized UI elements. A useful range of example projects were provided to augment PyMT's somewhat sparse documentation. Having established a presence with PyMT in the multi-touch community, the developers of the Kivy community appears to be growing.

The architecture of Kivy (Figure 3.3) includes support (lower left) for the external libraries typically used for images, text, audio, and video. Graphics (lower center) are rendered with hardware-acceleration implemented in C for performance. Support for many multi-touch input devices for different operating systems (lower right) is built in. The core providers section in the middle layer shows support for windowing, text (spell checking can be included), image loading, audio, and video. Many of an application's low-level details can be customized by configuration in the core provider section, permitting output using a 2D engine of your choice, such as pygame or SDL. The middle layer also contains API abstractions for OpenGL, offering access to advanced visualizations. The input abstractions allow for de-jittering and post-processing, and also serve as examples of simple event processing. For most applications the defaults in the lower and middle layers require no adjustment, however it is noteworthy that modifications at this level are relatively easy to achieve given Kivy's Python roots.

The top layer of Kivy's architecture is the one most developers will interact with, as this includes the application abstractions. The major distinction that the Kivy developers introduced after PyMT was the invention of the Kivy language, a set of declarative constructs meant to describe user interfaces and interactions. This definitional language will offer a critical foundation for potential visual designers of the kind we see exploited in more commercially supported architectures like Microsoft's WPF or Java Swing.

In addition to Kivy and PyMT, we must also mention another Python-based library, also with a declarative model at its core. This more mature library is called *libavg* [74]. Written in C++ for performance, the libavg project has been in active development since 2003 and uses Python as a scripting language to develop interactive media applications. It is notable for its use in engaging public exhibits, multi-user games, educational maps [75], and medical



(a) Simple Example Source: [62]

(b) Advanced Example. Source: [45]

Figure 3.4: Examples of visualization applications produced with Processing.

simulators. Libavg runs on Linux, Mac OS X, and Windows.

3.2.4 Java-based frameworks

Written in the Java language, the MT4j Framework [46] offers a favourable mix of architectural layers, UI components, performance, maturity, and extensibility. MT4j uses the language *Processing* [24] as its application container and as an animated canvas. Everything from GUI elements to touch keyboards are rendered by and listened to through this environment, and the event loop of Processing's PApplet is the heartbeat of an MT4j application.

The Java-based language *Processing* and its development environment were originally developed as tools for teaching computer programming to students, artists, designers, researchers, and anybody interested in doing computational work in a visual context. Processing runs on numerous platforms and has access to a great variety of libraries including (but not limited to) animation, 3D rendering, sound and video processing, networking, physics simulation, and numerous data import and export utilities.

At first glance, Processing can seem like a toy language (Figure 3.4a) but its apparent simplicity has made it attractive to interaction designers in several domains from network intrusion visualization to realtime geographic visualizations of telecom data (Figure 3.4b).

MT4j essentially extends Processing to support the essentials of tabletop interaction. It contains gesture definition tools, a physics engine, several sample applications, and reasonably good documentation. At the time we were doing the assessment for our project MT4j seemed a viable option, but as of 2015 the project appears to have gone dormant.

The architectural layers for the desktop package of MT4j are shown in Figure 3.5 (note that in this case the term 'desktop' distinguishes it from Android packaging; the actual surface could be a full wall or a large table display). As in the case with the Kivy diagram,

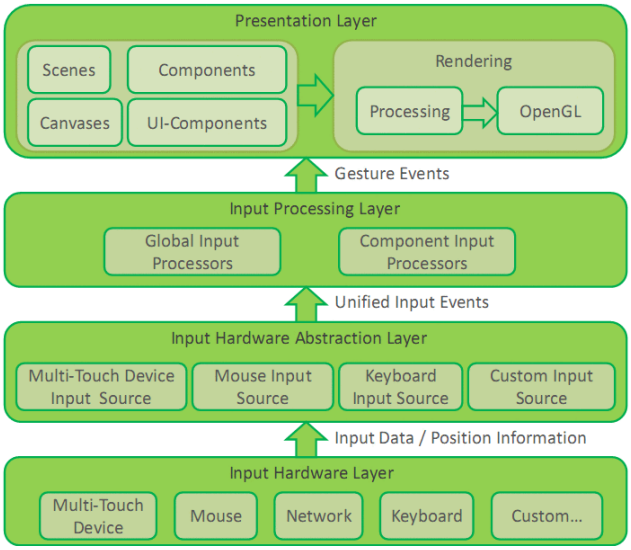


Figure 3.5: Multi-touch application MT4j architecture. Source: MT4j.org

the architecture is presented by the authors of the framework. Their focus in this diagram is somewhat different from that of Kivy. There are fewer options shown in the lower layers, and the middle layers explain the abstraction of events from the hardware layer. This reflects the somewhat more rigid design of MT4j, with low and middle layer decisions defined to support the rendering layer at the top right of the stack (Kivy lets you choose renderers appropriate to your platform). Note that the box called 'Processing' refers to the language and not a computational activity. A useful set of abstractions in MT4j that are somewhat less obvious in Kivy's diagram are the Scene and Component abstractions in the upper left of the presentation layer. These represent windowing abstractions, and the documentation for their use is quite good.

One aspect of MT4j's architecture that isn't clear from the diagram is the fact that the Processing language has a number of open source libraries of its own that developers might employ for a variety of input and output extensions. For instance MT4j comes with excellent demo programs including particle physics, 3D graphics, rich gestures, world maps, and many others.

3.2.5 Web-based frameworks

Recent advances in web browser design have raised interest in the web as a target platform for multi-touch. Visualizations have improved with the emergence of the HTML5 canvas,

the broader adoption of SVG, and the 3D potential of WebGL. Realtime communications have improved with the advent of WebSockets and WebRTC (Real-Time Communications). Performance has improved with advances in JavaScript compilation. Taken together these advances in browser design should make it possible to produce rich interactive and collaborative graphical interfaces for great varieties of user surfaces, from tabletops and desktops to tablets and smartphones.

The advantages of web deployment are that one application can be run on any platform, and without any need to explicitly install or update. The disadvantages are limited access to local sensors or resources, but those same limitations are also beneficial for security. The basic idea is to leverage web browsers as standards-based development platforms, and improvements in browser performance and capability make this increasingly attractive. It must be mentioned, however, that browser implementations are not entirely uniform. The touch specification for HTML is still evolving, and browser compliances vary. We find a proliferation of client-side JavaScript libraries for each of the major architectural components outlined in previous sections, and many of these are excellent. But still there hasn't been a single reference web framework for multi-touch that includes *all* the important pieces, at least not yet.

In the past it might have been worth considering Adobe Flash as a potential development platform for web-enabled rich user interfaces capable of touch, but with Apple refusing to support Flash on iOS, with Adobe's recent announcement that after version 11.2 they will be discontinuing their standalone player for Linux (current plans are to make it available only as a plugin for Google Chrome), and with Microsoft's move in Windows 8 toward a "plugin-free" version of Internet Explorer, but at this point HTML5 and JavaScript have more cross-platform merit.

Moving away from Flash we find no shortage of pure JavaScript visualization toolkits that are great candidates for web-based surface interaction. The Processing language used by MT4j has been ported to JavaScript [25], and with the addition of touch (in release 1.1) it is now possible to reproduce something like MT4j's interaction capabilities in canvas-capable web browsers. The D3 library [8,9], Raphaël [5], and KineticJS [43], also handle touch events.

Choosing a framework may depend on your application and any legacy browsers you may need to support. Processing.js and KineticJS are both based on the HTML5 Canvas, while

Raphaël and D3 manipulate SVG images (note: Raphaël quietly produces VML images for older versions of MSIE). Canvas-based images may be quicker to animate, particularly when adding or removing large numbers of distinct elements. SVG-based images may be easier to manipulate and they scale exceptionally well.

While the SVG protocol has been specified for some time, it has only recently begun to have a broad impact. Better built-in browser support is one reason, and now more usable JavaScript libraries, such as Raphaël and D3, are gaining widespread use in visualization applications. Ironically, the inclusion of the raster-based HTML5 Canvas appears to have drawn attention to the vector-based SVG as a useful alternative more suitable for applications requiring scalability. The lack of support for Adobe Flash in many mobile devices may also be a factor.

One of the challenges to overcome when dealing with multi-touch is the need to unify gesture processing across multiple computing platforms. Each of the major operating systems – particularly in handheld devices – has its own methods for producing and handling advanced gestural events, and browsers themselves have differences in how they consume these events. Work is under way by producers of open source JavaScript libraries to simplify this situation and offer a common gestural vocabulary [72].

3.2.6 Choosing a framework

How to choose between the different possible toolkits will depend on a number of factors. Issues include your experience with programming languages, your comfort with image manipulation, the availability of touch-based hardware for development (perhaps needing less expensive simulation) and release (potentially targeting a variety of mass market devices), the range of platforms you need to target, and the need to incorporate novel modes of interaction.

Many of the open source toolkits are rapidly evolving projects with open issues on the various operating systems they support. One may find that one's new idea for multi-modal interaction violates a number of assumptions that span an entire open source architectural stack. This could be seen as a barrier to research, but it can also be seen as an opportunity to improve an open source toolkit or even as an opportunity to research and derive your own tools.

In the large display and mixed-surface environments – our own current focus – we found

that both the Python-based and browser-based environments were a good fit. Kivy developers have targeted the broadest range of surface platforms, from large custom-built displays to smaller iOS and Android devices, but now we are finding a range of native build processes for web-based frameworks as well.

The browser-based tools are an ongoing development challenge but we see great potential in its near-instantaneous deployment and its use in remote collaboration with small and large surfaces. Our proof of concept employed node.js [40] as a web server, a collaboration server, and a TUIO interpreter, and we used client-side JavaScript to encode a gesture processing layer using a state-machine model. We chose an SVG-based visualization toolkit for the advantage such tools bring in designing interfaces. For example, it is possible to sketch a visualization in Inkscape, export the resulting SVG code, and then bring it to life with Raphaël or D3 [73]. Detailed JavaScript debugging is made possible even in remote handheld devices with the advent of remote web inspector tools [53].

Of course many people work within a Microsoft environment for development and deployment, and use tools based on Microsoft's .NET Framework. For instance, a research framework for "proxemic interaction" at the University of Calgary [50] is published as an open source .NET library. Also Microsoft's *Expression Blend* and *Expression Web* designers have been designed to work with the Surface 2.0 SDK. Designers can find numerous videos explaining how to use these commercial tools to produce professional designs for Windows-based systems.

3.3 Issues arising

To conclude we summarize issues that arise when selecting multi-touch frameworks to enable multi-surface collaboration (collaboration that is enabled by multiple surfaces).

Few multi-touch surfaces have the means to identify the source of gestures when several collaborators are interacting with a single screen. Some notable exceptions are the Diamond-Touch [20] table which identifies users by means of capacitive charges in their chairs, or systems designed around Anoto digital pens which have unique Bluetooth identifiers. Other techniques have been suggested using a variety of heuristics, such as differences in territory, skin tone, or behaviour. However, the ubiquity and low cost of tablets and smartphones offers a excellent opportunity to provide the identiy of collaborators, as well as additional

modes of interaction beyond touch. Further, tablets and smartphones as additional devices for collaboration can offer opportunities for private exploration, offline data manipulation and preparation, and interactions requiring personalization or authority.

The technologies mentioned in the previous section, particularly web-based frameworks, are a good starting point for collaborating across multiple devices. However, there remain differences in how gestures are shared with browsers within each of the main handheld operating systems, and so it may be worth designing for a mix of browser-based and native code.

Prior to exploring browser-based technologies we experimented with multi-screen approaches using MT4j and PyMT. Our early designs were based on simply sharing TUIO touch events across multiple collaborating systems, and we quickly discovered a number of challenges in this space. These challenges were, briefly:

- a shortage of published research in the domain of simultaneous multi-device multi-touch input
- a lack of research on gesture interpretation in the context of collaborating devices
- system and protocol implementations that assumed only single-system sources of touch

Multi-device touch input was a peripheral component of our larger projects and while we did proceed to solve the essential implementation issues with MT4j there remained a number of challenges to be addressed. The shortage of published research was addressed in a paper published months later by Liu, Feng, and Li [48] that presents networking protocols and algorithms that ensure responsive and error-free streaming of multi-touch events between networked devices.

Additional work is still required in the social aspects of this component of collaboration. Our own experiences did manage to generate some potentially useful insights. We offer the following observations:

1. Remote cursors are very useful for collaboration, however they should probably be for display only on remote machines. Multi-touch gestures should generally ignore remote cursor sources.

2. In the event that an event requires multiple collaborators, for instance in a voting scenario or in cases where more than one participant is required for authority, then alternative schemes can be devised that make use of shared system state.

3. The challenge of sharing application state across multiple devices gives rise to an important question of the identity of objects. It is important to draw appropriate distinctions between actual objects and inferred or proxy objects. Mutability (the ability of objects to be changed) of shared objects must follow logic that meets mutually shared goals of participants.

4. Remote manipulation of shared visual artifacts can create conflicting events. Depending on system latency, remote users may consistently have their updates rolled back. All displays should quickly reflect provisional changes to shared state.

5. Display of shared state on client devices may represent an aggregation based on large amounts of underlying data, however it is impractical for each device to keep local copies of each of these underlying objects. Care must be taken to provide enough copied data for responsive local manipulation but not so much that its maintenance becomes a performance burden.

6. Additional performance cost on small devices can come from the container classes used to render objects on screen. This is particularly evident when displays use simulated physics for animated realism. Since the screen itself is physically limited this forms a natural opportunity to improve performance by reusing existing containers that have scrolled outside the display area rather than recreating them as manipulation demands.

Chapter 4

Architecture

This chapter describes our efforts to support collaborative multitouch visualization using web technology. The shift away from Python and Java to web-based tools was motivated by several factors, among which were:

1. distributed architecture for mixing co-located and remote collaborators
2. ease of deployment to facilitate agile iteration of evolving designs
3. improved support for high performance web applications
4. reduced effort in cross-platform support for desktop and mobile devices

First we describe TuioHost, a plugin-free port the Tangible User Interface Objects (TUIO) protocol that enabled us to provide multitouch and other rich inputs across all available web browsers.

We describe an architectural proof-of-concept using TuioHost to explore the feasibility of a client-server web application architecture using various popular open source tools.

We then move onto the architecture ultimately adopted. Armed with our multitouch tools and the new visualization knowledge from our proof-of-concept, and keenly aware of the schedule risks with a single developer, we discuss a revised architecture that takes advantage of the new productivity offered by (at the time) recently announced web development frameworks.

4.1 TUIO Host

As of this writing of this thesis (August 2015), there remain numerous outstanding issues with desktop browser support, a state of affairs that is often compounded by inconsistent gesture detection across operating systems. We wrote the TuioHost library [80] to sidestep these inconsistencies, a decision that we thought would be temporary but which has turned out to be valid until today.

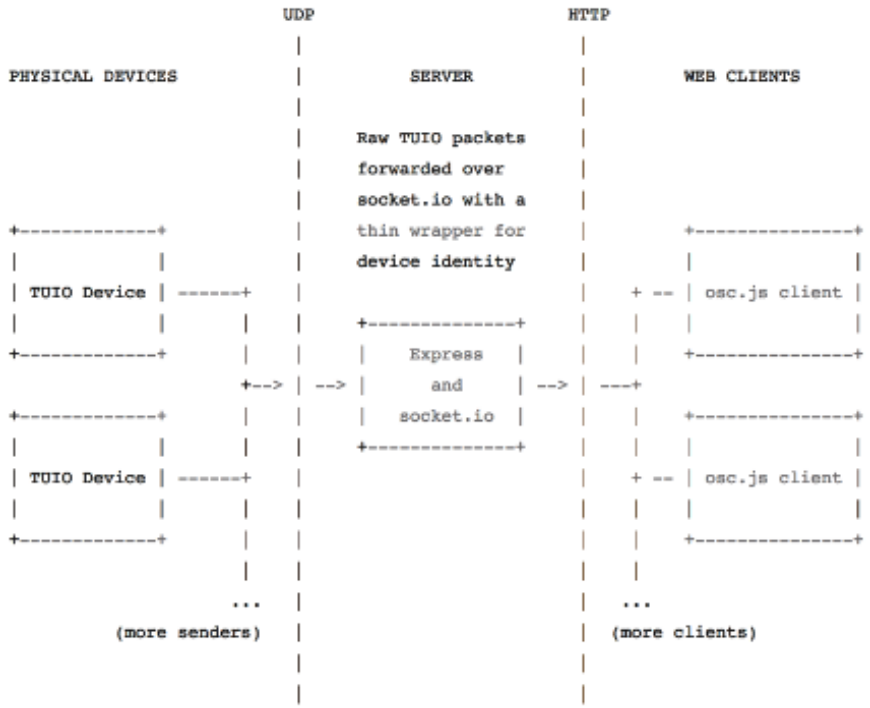


Figure 4.1: TUIO Express

The TUIO protocol (see Section 3.2.1) is a UDP-based event messaging standard that is well supported by multitouch devices. TUIO adopted as its message structure the Open Sound Control [15] protocol (OSC), which was designed to bring modern computer networking to musical instruments and control panels. The use of a simple network protocol made it possible for us to incorporate multitouch events from multiple devices, and to consider novel ways of combining them into individual or collaborative gestures.

Browsers do not natively support TUIO. At the time we wrote TuioHost it was possible to install an NPAPI plugin TUIO bridge for Firefox, but we learned that it would soon be disabled in other browsers due to security concerns with NPAPI [30]. We chose to develop a bridge for TUIO events using NodeJS [40]. This strategy permitted us to avoid plugins altogether (unless you count NodeJS as a kind of plugin), and it helped eliminate some of the problems with early gesture capture by host operating systems.

TuioHost and our recent rewrite called TUIO Express [81] (an updated version that simplified the server and pushed processing of TUIO packets into the client, see Figure 4.1) demonstrated the viability of a TUIO implementation in the browser, as well as support for

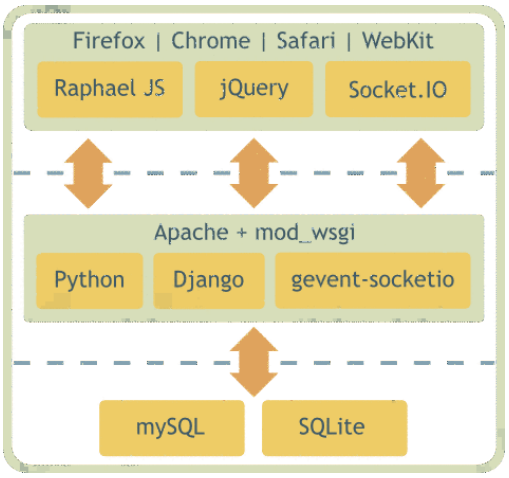


Figure 4.2: Web multi-touch application architecture.

multiple distinct TUIO sources through a single server. This enabled our development of tools from large surface displays as well as from tablets based on Android and iOS. It also provided support for event types still not supported in browsers today, such as those from 3D sensors and tangible user interface elements.

The TUIO-Express server performs the following roles:

- provides client-side static resources (JavaScript and HTML)
- listens for raw TUIO bundles over UDP
- wraps bundles to add sender identity (many devices lack "source" packets)
- forwards wrapped UDP packets over socket.io

The browser-based clients perform the bulk of the work with respect to TUIO:

- interprets raw binary into JavaScript-legible bundles
- maps OSC bundles into packets and then infers operations from packet types
- maintains cache of TUIO objects by sender and type
- retires defunct objects no longer appearing in 'alive' packets
- raises events for downstream applications

4.2 Proof of Concept

The architecture diagram in Figure 4.2 shows our browser-based client-server architecture supporting desktop and handheld browsers. This architecture formed the basis of our first

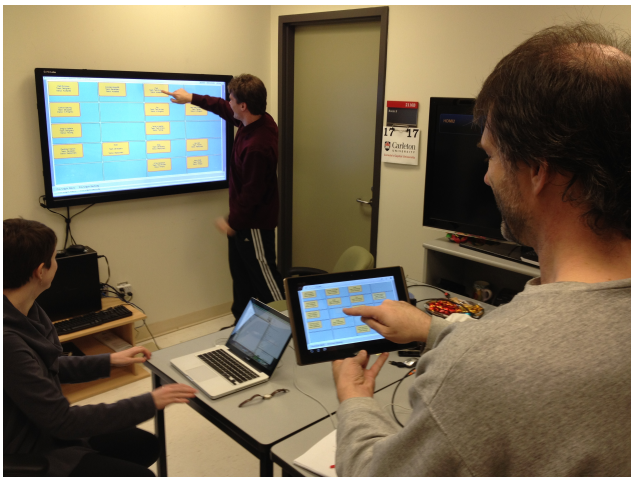


Figure 4.3: Proof of concept Digital Cardwall.

proof-of-concept application, a demo cardwall using this architecture. Figure 4.3 shows this application updating shared state across a large surface display, an Apple MacBook, and an Android tablet.

Note that the ability to share a display across the web brought with it a risk that latency will affect shared application state. With multiple users potentially contributing multiple cursors (or touch-points), questions might arise as to which input events to process, how to present interim or provisional states, and what to do when conflicting updates occur. While these were all interesting technical challenges, we were primarily interested in the co-located use cases and so we resolved to treat potential latency problems as edge cases that would require user decisions.

4.3 Revised Architecture

There are numerous potential uses for such multi-display environments, particularly in real-time collaboration. Rapid prototyping and deployment of new interaction methods that employ multiple displays can become constrained by available architectures, particularly when bringing these prototypes to users' workplaces for validation.

Meteor [1] is a web application development framework that fosters collaborative real-time interaction. It also sets out to improve the developer experience for database driven single-page applications by using the same semantics on the server and in the browser. We discuss Meteor in detail below.

We knew that we would have only a few months to develop the full prototype for ACH-W. While our proof-of-concept was a success from the standpoint of demonstrating the required touch-based interaction with SVG visualizations, we determined that the development with the custom architecture discussed earlier risked taking too long. With only one developer producing the many features that would ultimately be required for ACH-W we decided to take the lessons learned from our cardwall and fit them into a more mature web development framework.

The Meteor team hopes to define "A better way to build apps", an ambitious and laudable goal. Their product was still pre-beta at the time we started using it, but on first examination it appeared to be in an advanced enough state for use in prototyping. It also fit well with our previous experience with the network application platform Node.js [40].

Our revised architecture also made use of a popular JavaScript visualization library called D3 [8] for visualizations. The name D3 is derived from the phrase "data-driven documents", an architectural concept that fit very nicely with Meteor's automatic refreshing of web clients subscribed to shared and persisted data. We describe this connection in more detail in Section 4.4.2.

4.4 Architectural Rationale

The last decade has witnessed a gradual and perhaps reluctant march of web application logic from the server to the client [7, 23, 71, 76, 84]. The reluctance to migrate application logic entirely to the browser can be explained by a number of factors, such as a lack of defined application standards, concerns about security, and the consumption of battery, memory, and expensive bandwidth on mobile devices. These issues remain relevant today, but there are promising developments in open source communities forming around HTML5 and Node.js.

Web developers have historically been saddled with the challenge of forming mental models for their applications out of assemblies of nested HTML tags that bear little resemblance to application structure. This conceptual *impedance mismatch* has often been applied to *object-relational mapping*. In HTML we find another mapping of objects into attachment points like DIVs and SPANs, a conceptual leap alleviated only slightly by CSS class names.

The reason for this may be that prior to HTML5, web browsers complied with stricter rules when rendering HTML into the web pages we enjoy. It was possible to create the illusion

of a web "application" with JavaScript libraries such as jQuery, but these applications had to be defined in an *imperative* fashion that waited for rendering to complete before "tricking" the document object model (DOM) into behaving like an application. The problem with this approach is that the development effort seems to scale exponentially with respect to application complexity.

The HTML5 specification loosened some of the constraints on HTML *tags* (keywords that declared segments of HTML like `<HEAD>` or `<BODY>`), inviting a new generation of JavaScript frameworks to modify the DOM earlier in the rendering phase. This allows for more *declarative* coding techniques that define web applications with *data binding* in client-side templates.

The advent of client-side templates allows the developer to treat their application as an assembly of rich *custom elements* (note: we are using a somewhat looser definition of this phrase than you might see in the upcoming W3C Web Components specification [2], in that we include *any* new HTML tags defined by the programmer, not just those that meet the upcoming spec). Frameworks like Meteor and AngularJS [29] encourage coherence around *domain* semantics rather than around units of HTML rendering. Custom elements then become much easier to conceptualize as logical *views* with reactive binding for data and events.

Meteor takes an additional step toward improving the conceptual framework of web applications by simplifying access to data. Their reference implementation describes database queries in a variant of MongoDB [37] syntax and it offers the same query methods in both server and client JavaScript code. Meteor clients keep a subset of the server data using *publish-subscribe* semantics, creating a "MiniMongo" cache (see Figure 4.4) that can be easily redefined with a key-value pairs stored in the client's Session data. Templated HTML fragments are bound to these cached document collections that allow both reading and writing. Subject to permissions that are easily defined, cache writes are flushed to the server where they are automatically replicated to other subscribers.

Meteor's real-time replication uses their own Distributed Data Protocol (DDP) to maintain synchronization between the server and its connected clients. It is worth noting that Meteor's default web client is but one possible implementation. In addition to independent JavaScript clients, there are working implementation projects [51] (some already dormant) for fifteen different languages (e.g. C#, Go, and Haskell) and platforms (iOS, Android,

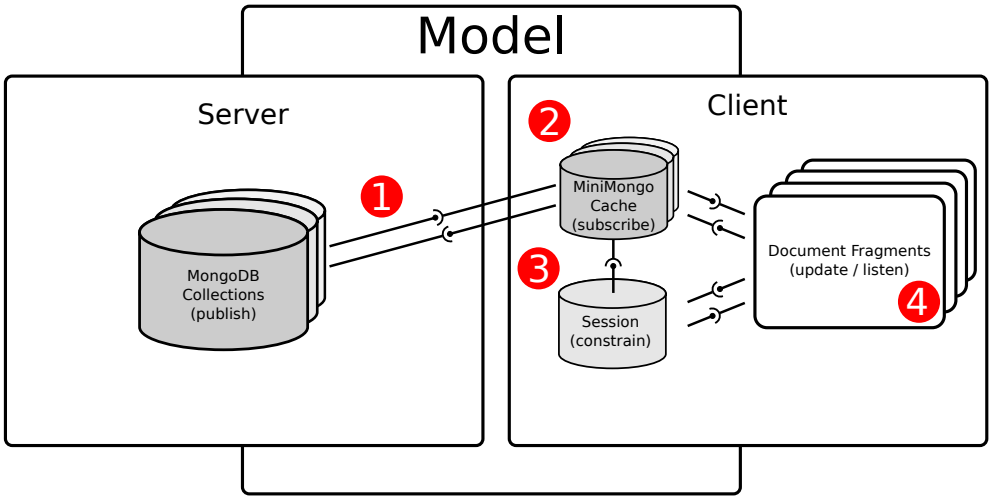


Figure 4.4: Conceptual Data Model using Meteor.

.NET). The potential for this degree of client-side flexibility was apparent early in our choice of Meteor, though it is only recently that this potential has been realized.

4.4.1 Data Flows and Distributed UI

With its transparent replication, Meteor is able to define a *reactive* application architecture driven primarily by database updates and user events. Database changes–from both the client and the server–are managed with the concept of *reactive computation*. This architecture is similar to traditional client-server MVC web frameworks in that *views* still act as *observers* of a *model*, but Meteor differs in that it shifts the architectural elements of *controller* and *model* away from the server and into the client. Meteor transforms the client-server relationship into more of a cache maintenance exercise, improving user perceptions of performance by permitting them to update model values locally, subject of course to potential correction by the server as required.

Another difference between MVC and Meteor is in the location of view composition. Client-server MVC architectures construct HTML views by means of templates on the server, and the more advanced frameworks update these views by means of AJAX [27] client requests to avoid full-page refreshes. The most advanced MVC frameworks take the additional step of refreshing views with the newer "Comet" approach [64] for server-to-client updates. This

creates a tight coupling between server and client, since the server must know something about how views are structured. In Meteor there is no such coupling, since the templates are delivered to the client on application startup, and view construction is handled entirely in the browser. This has advantages in both performance and flexibility, though it creates a need for additional checking on the server to ensure that all updates are authorized and valid, tasks that are well supported by the framework.

Meteor templates are defined with a small vocabulary of substitution, condition, and repetition constructs. Meteor performs an analysis of these features to label fragments of a web page as *reactively* dependent on changes to data, creating bi-directional bindings between the DOM and the local data as necessary. The local data in turn is updated via DDP as mentioned above. Developers will typically use template logic to react to the various events arising from model changes, or they can define new queries based on custom logic (e.g. reacting to a timeout condition).

Meteor automatically replicates data to all subscribing clients using a mechanism described as server-published collections (note that it is possible to have client-only collections as well). Figure 4.4 shows how this allows for a conceptual model of data that transparently spans the client and server contexts. The key lies in applying reasonable subscription parameters to collections published on the server (label 1 in Figure 4.4). Subscribing to massive data collections can be achieved by defining appropriate filters, limits, and database indexes. This ensures that the data replicated to subscribing clients (label 2) appears in reasonably small quantities that can be quickly updated and rendered. Subscription parameters are typically kept in Meteor's client-only Session store (label 3). Document fragments bound by template semantics (label 4) will be automatically re-rendered when either the collections or the session variables are changed. Note that changes to session variables are private to the client, whereas changes to collections can come from the client or the server.

The automatic replication of published server and client collections (subject to filters of course) allows for multiple screens to be updated nearly simultaneously, and it is this feature of Meteor that lends itself to distributed user interfaces. If two or more users wish to see precisely the same subsets of shared collections, it is a simple matter of defining their subscription parameters in a common collection, rather than in session variables. By adding an additional field to shared subscription data it is possible to partition these subscriptions

into multiple collaborative contexts, for instance defining metaphorical *rooms* or *lobbies* where all users within these spaces share common streams of domain data events.

4.4.2 Data-Driven Visualizations

Thus far we have described a fairly typical scenario for HTML development, but the use of templates and data binding can also apply to rendering SVG. Notice the template placeholders on lines 3 and 4 below.

```
1 <body>
2   <svg id='map' >
3     <g id='countries'>{{> Countries}}</g>
4     <g id='circles'  >{{> Circles}}  </g>
5   </svg>
6 </body>
7
8 <template name='Countries'></template>
9 <template name='Circles'></template>
```

Listing 1: Simplified usage of a Meteor template using SVG

Listing 1 shows a sample template definition for drawing with D3.js within Meteor. The empty templates on lines 8 and 9 are linked to JavaScript objects such as the one being extended in Listing 2. The use of two templates within a single SVG drawing gives us distinct execution contexts that prevent unnecessary refreshes of relatively static data sources. For example, the GIS data to draw the map in Figure 4.5 will not be changing, and so should not be re-rendered, whereas the event data represented by circles definitely should.

To continue this example, line 1 of Listing 2 defines a data collection. This collection is queried on line 14, forming a reactive dependency that Meteor automatically updates bi-directionally. Line 3 shows that we are defining methods for the template corresponding to Line 9 in Listing 1. Lines 16 to 25 define what to do when new objects enter and exit the client-side collection. Configured as shown, the SVG will simply insert events (drawn as circles) when events appear in the data, and then will remove the appropriate circle when its event disappears from the data.

This example is greatly simplified to conserve space, and the simplification undersells the power of D3.js. The example is taken from a visualization of Apache web server log events we developed for a course project in usable security. In the actual code used to

```
EventData = new Meteor.Collection("events");

_.extend(Template.Circles, {
  rendered: function () {
    var self = this;
    if (!self.handle) {
      self.handle =
        Meteor.autorun (function () {

        var svg = d3.select('#map');

        var circles = svg.select('#circles')
          .selectAll('circle')
          .data(EventData.find().fetch());

        circles.enter()
          .insert('circle')
          .attr('cx',
            function (d) { return d.x; })
          .attr('cy',
            function (d) { return d.y; })
          .attr('r',
            function (d) { return d.r; });

        circles.exit().remove();

      });
    }
  }
});
```

Listing 2: Binding database change events to visualization updates.

generate Figure 4.5, its functions corresponding to Listing 2, lines 19 and 21 were defined to derive a circle's coordinates from a function applying a geographic projection, and the radius calculation in line 23 applied a logarithmic scale for intensity. Additional code was used for tooltips as well as mouse clicks or screen touches. The map was also configured to support zooming and panning.

Web pages can be composed of multiple nested templates. Figure 4.6 shows an Apache Log Dashboard composed of four parent templates with several children. This dashboard queries a database of approximately a million log entries using Schneiderman's visualization mantra "overview first, zoom and filter, details on demand" [66].

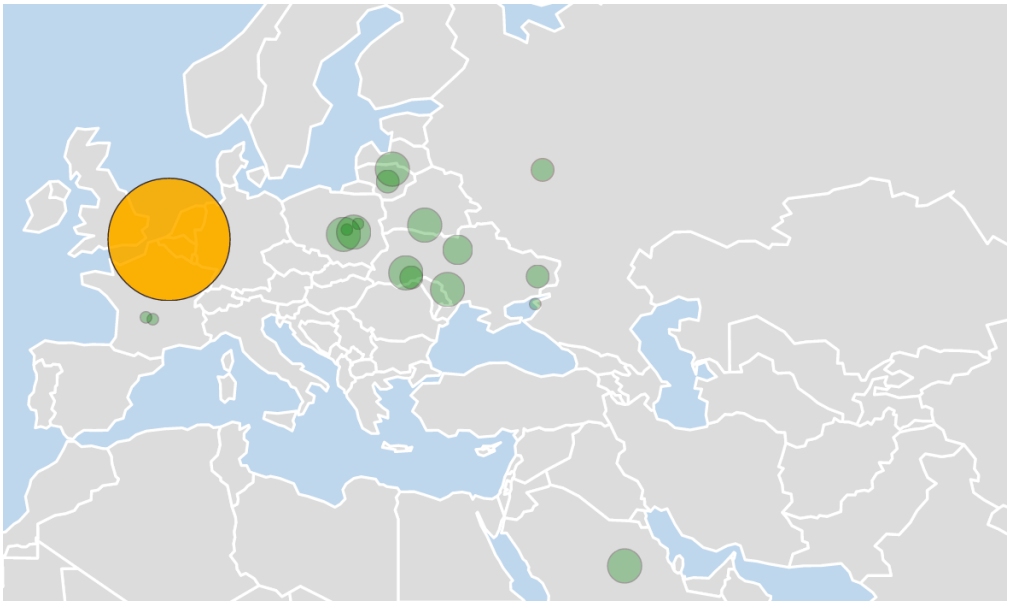


Figure 4.5: Using geographic projections in D3.js

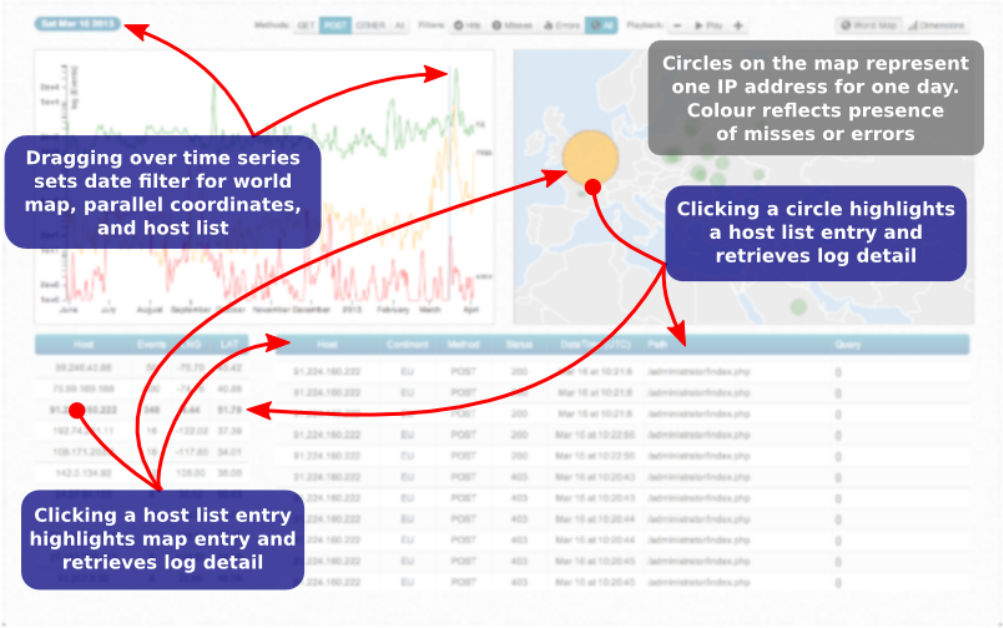


Figure 4.6: Apache Log Dashboard

4.5 Demonstrating Design Flexibility

The original design of our web log application called for a single user on a single screen. As an exercise to demonstrate the productivity value of this architecture, we wondered how long it would take to put the world map on its own page in such a way that a remote user could listen for session filters applied by a user on the main dashboard. The original design kept session changes driven by touch or mouse events private to the dashboard user. A new collection was defined called *SharedSession*, and the time series view and the full-page world map template were modified to share subscription parameters through this new collection. There were some additional changes to allow routing by URL using Backbone.js [21]. We note, parenthetically, that Meteor's early beta did not ship with the ability to route to a new page based on a URL, and so we needed an additional library. This had the virtue of solving a problem and demonstrating that non-Meteor JavaScript libraries could be used to good effect. Back to our challenge, in less than an hour we had deployed a distributed near real-time user interface. To have achieved a similar refactoring of our original proof-of-concept architecture we would have needed several days.

Figure 4.7 shows how an application written using Meteor can be quickly partitioned across systems; a dashboard originally designed for a single analyst was split into a distributed system with twinned views in under an hour. Both displays are fully interactive and are kept in sync by Meteor.

4.6 Preparation for Further Applications

A design challenge with one test cannot be considered complete validation of our hopes, but it did demonstrate that Meteor could easily handle our needs for an easily composed web UI with custom real-time capable visualizations. It was also capable of supporting writes to shared data models from multiple collaborators. The term the Meteor team uses for this feature is "latency compensation", a phrase that sounds like it solves a different problem that we chose not to explore in great depth, namely that of distant users losing out on races to the same resource. Meteor doesn't currently compensate for unequal latency either. For now all updates take place on the quite reasonable assumption of "last writer wins".

In the short term we decided to ensure we used graphical cues to help distant collaborators

Figure 4.7: Dashboard Distributed in 42 Minutes

cope with this assumption when competing for the same on-screen elements. The sharing of real-time data should at the very least help with advisory information, and pushing what is in many cases a human coordination problem into the users' hands is an important first step. In the longer term we might explore more automated approaches to the management of unequal latency.

Many of our early design experiments focused on large screens with multitouch interfaces, and the combination of Meteor and D3 promised to help with the server infrastructure in these plans. Some challenges remain with respect to gestures, in particular gestures that require time to resolve or that can be interpreted in more than one way. Meteor itself will not solve these issues for us, but experiments with plugins suggested their flexible package management would provide support for any new libraries we might write or import. The functional programming style made possible by JavaScript is a good fit for gesture handling, and in another early experiment with a transparent SVG overlay offered a useful surface for capturing, interpreting, and forwarding gesture events.

Also outside Meteor's purview but definitely within our own is the issue of how different browser/OS combinations consume gestures before passing them to our applications. This was an interesting challenge that would not be encountered in a native environment, and future work will allow us to address this problem with native builds using the DDP ports and native development tools, or by using products like Apache Cordova to generate handheld apps.

Chapter 5

ACH Walkthrough: Design and Implementation

5.1 Introduction

This chapter presents ACH Walkthrough (ACH-W) [82], a prototype software application to demonstrate the potential benefits of surface technologies in collaborative intelligence analysis.

Our general approach is the application of surface technology to collaboration, and especially analysis work [11]. In this, we emphasize the potential for surface technology to support complex collaborative work.

5.2 Field Study

The ACH-W project stems from a field study of intelligence analysts at work [10] where we were privileged to observe a series of sessions with professional intelligence analysts in a realistic setting. Our goals were to gain an understanding of the natural forms of co-located collaboration present within analysis teams with a focus on their current use of available artifacts (e.g. flipcharts, paper, post-its, desktop computer displays, projected displays, etc.), and to consider how large interactive surfaces might help support these analysis efforts.

The setting and equipment for the field study was provided by their employer, a government agency who was interested in the potential benefits of technology to support co-located collaboration. The study environment was designed to simulate their actual work environment to the degree permissible within necessary legal constraints. There were ten analysts in the study, three of whom also had software development skills. One of the ten analysts played the role of an internal client, and another volunteered for the role of facilitator. The remaining participants were presented with a significant cognitive and technical challenge, wherein they were given four non-consecutive days to analyze a corpus of half a million emails generated by the employees at ENRON Corporation in the 3.5 years of financial irregularity

immediately preceding their bankruptcy [18]. The participants were tasked with producing a "bowl of knowledge" for the client, which is a collection of analyses across a range of potentially relevant topics. While a few participant choose their topics entirely from the corpus of emails or from 'open source' information relating to ENRON's case available the Internet, most participants drew from both.

The process unfolded in six steps, the first four of which took place in the first half day. The fifth step lasted approximately 2.5 days, and the sixth step was the last half of the fourth day. The analysts worked in an open office setting, with individual workstations separated by low partitions, and with two open areas used for group meetings. Their level of collaboration ranged from whole group agile standup meetings, through small group brainstorming sessions and paired work, to entirely solitary time.

1. Reading of background documents (solo)
2. Brainstorming issues (three groups)
3. Selecting best issues (three reformulated groups)
4. Selection of analytical technique (same groups from 3)
5. Actual analysis (solo, with collaboration as required)
6. Presentation of 'bowl of knowledge' to client

Up to three observers were available to shadow the analysts to observe the exercise (I was present for the first and third sessions). We made our observations using paper notes and photographs of artifacts, and our observations were coded according to the duration and nature of the collaboration. The nature of the collaboration was based on an Activity Theoretical model articulated by Engeström [22] that was extended to capture features distinct to this particular study group (Engeström's original three forms had come from courtroom activities, and they did not capture certain forms of disjoint behaviour observed in our study). These forms were identified as:

Coordination — When people 'coordinate their work' each person has a distinct objective and each follows a subtly scripted role ... which is tacitly understood by others. [22]

Cooperation — When people 'cooperate' they share common problems and a common objective while following scripted roles. [22]

Reflective Communication — When people 'reflectively communicate' they reconceptualize their own organization and interaction in relation to their shared objectives,

and stop following a script. Reflective communication occurs when tensions build and normal work is re-evaluated and developed. This may be to re-assess objectives (the attributes of analytic outcomes) or to adjust other aspects of the activity. [22]

Side-by-Side Coordination — When an analyst was working side-by-side with others in a shared space, but on their own analyses, we called that 'side-by-side coordinated' work. This is because the team as a whole coordinated their actions, i.e., behaved in a way that made side-by-side individualized work possible (mostly by working quietly). [10]

Side-by-side Cooperation — When an analyst was working side-by-side, but was doing something for another analyst, as was often the case with the developer-analysts, we called that sort of paired work 'side-by-side cooperative' work. In this case an analyst and a developer-analyst pair (rather than the entire team) were, for a duration, jointly aligned on a singular objective, which was the analyst's current technical issue. [10]

A number of findings related to availability and nature of physical and software artifacts and tools, such as how a lack of convention in representation in the brainstorming phases led to a loss of information (i.e. inconsistent semantics of colour choice and physical arrangement in the use of post-it notes), how the use of a poster in a morning standup meeting early in the analysis phase led to a rich and productive discussion (notable for its absence and the concomitant absence of discussion from other teams), how the lack of support for collaboration in certain software tools had a negative impact on the productivity of two analysts.

Since it turned out that most analysts were engaged in similar activities at the same times, we were able to chart an estimate of the strength of collaboration of the entire group. Figure 5.1 represents the strength of collaboration (vertical axis, higher is more cooperative) of the entire team over the four day session, where the days are delinated by dotted lines and the phases or steps are delineated by red lines. The dots indicate where a transition from one type of collaboration to another took place. We note that true collaboration (two or more people directly interacting to achieve a common objective) is found only on the top two points on the *y-axis*. The lower three forms were essentially disjoint behaviour where analysts worked separately.

We observed that the bulk of the *cooperative* collaboration in step 5 (the longest step — see Figure 5.1) came as a result of the technical challenge of working with the structure of the data format, where we saw the programmer-analysts cooperating with other analysts

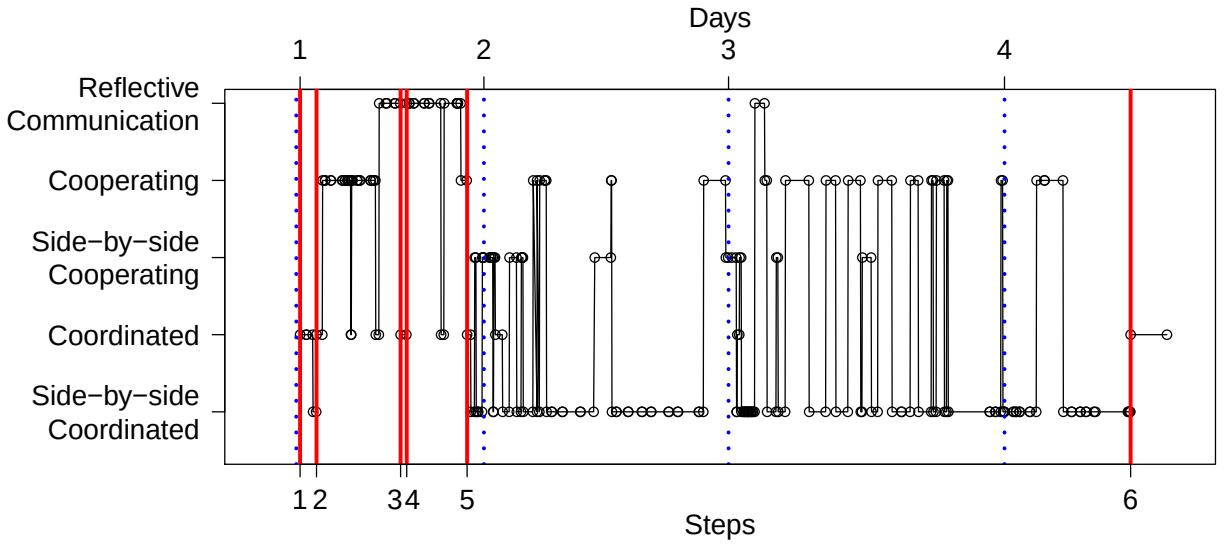


Figure 5.1: The process of collaboration across time. Solid lines show steps in the analysts' process. Dashed vertical lines show days. Note the absence of reflective communication in step 5, selecting & analyzing issues. Source: Brown et al. [10]

in enabling data extraction. We were surprised to find that a distinct shortage of *reflective communication* during step 5, in spite of the evidence that software developers regularly use this form of collaboration to improve development outcomes. We also observed a lack of tightly coupled analytic work, which again could have offered the benefits of reduced errors and increased productivity, as it does in software teams [79].

5.3 Establishing Requirements

As part of the field study, we observed two analysts performing an "Analysis of Competing Hypotheses" (ACH). The ACH technique was developed by Heuer [35]. This process first encourages consideration of multiple hypotheses that may explain a set of evidence, and then emphasizes disproving and eliminating hypotheses using diagnostic evidence. Scores are computed for each hypothesis to indicate the degree to which it has been disproved.

Our early studies suggested that ACH would benefit from collaborative support because the consideration and judgement would both be assisted by team discussion. We found calls for increased collaboration by authorities in the intelligence analysis world. Heuer and Pherson suggest that their collection of structured methods [36] can support collaboration

and reduce cognitive bias. Hackman [31] concurs and emphasizes that collaboration both improves outcomes and contributes to the development of individual and team skills.

We reviewed other versions of ACH software, namely the version developed for individual analysts at PARC [59], and two versions designed for collaboration, namely Globalytica Think Suite's Team ACH [28], and Open Source ACH [12]. We created extensive requirements for a collaborative version of ACH using surface technologies. The main requirements were that:

1. A collaborative version of ACH should enable part of a larger process where analysts alternate between individual work on an ACH and collaborative work on an ACH;
2. Analysts should be able to easily view evidence documents while working on an ACH analysis, and we speculated that a mixed-display environment would support this best;
3. Collaborative ACH work should be enabled by a walkthrough process whereby members of the team take on roles that would strengthen the analysis, while they walked through all aspects of the analysis and checked or extended its content.

We have mainly focused on requirement 3. We saw the walkthrough support as an important part of the tool, given evidence that a fair number of users were new to ACH. Our requirements also introduced both a new collaborative practice as well as a surface application. Together these would aim to improve an ACH analysis by enabling face-to-face discussions about the attributes of the analysis, e.g., its completeness, its correctness, and so on. Our application software accomplishes all these goals as a functional prototype. Figure 5.2 shows the software in use. The data set we use to illustrate the software is from publicly available material to investigate the collapse of ENRON Corporation [18].

5.4 ACH Glossary

Before we describe the walkthrough steps in detail, we must define some of the terms coming up in the ACH process.

Hypothesis: Any testable proposition. This could be an explanation of a past event, or a projection into some likely future. It could be a medical diagnosis, or a claim of flight risk on the part of the criminally accused. The entire list of hypotheses for an analysis should be as comprehensive as possible and mutually exclusive.

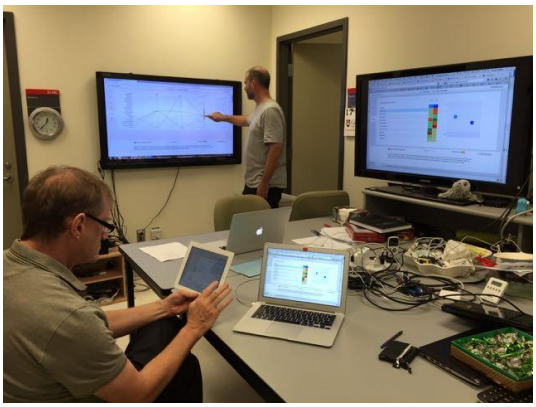


Figure 5.2: ACH **Walkthrough in use:** Running with synchronized data on a large multi-touch screen, on a laptop computer, and on a tablet.

Evidence: A finding that might pertain to the analysis. This is deliberately vague, as it includes virtually anything that would advance understanding of the truth, including speculation, rumour, and even *conspicuously absent* evidence, particularly if this absence is strong diagnostic (e.g. absence of troop buildup when analyzing the likelihood of imminent attack, or an absence of skin rash when ruling out measles).

Relevance: Pertaining to evidence, analysts are asked to rank an item of evidence for its contribution to the overall analysis (low, medium, or high). This evaluation is meant to reflect an item's evidentiary contribution to the entire question at hand, not each hypothesis (that's done separately, in the consistency rating). Reasons for low relevance are left to the judgement of the analyst, but can perhaps be illustrated with a police example: circumstantial evidence might be ranked 'low' (e.g. *footprints other than victims' were found*), where evidence determined by forensics might be ranked 'high' (e.g. *plaster cast of the shoeprint matched the suspect's footwear*). Relevance is used as a weighting factor for consistency ratings.

Credibility: Again pertaining to evidence, analysts are asked to give their opinion of the credibility of the claim (low, medium, or high), taking into account its source (i.e., are they trustworthy), its overall likelihood, the potential for tampering or deception, etc. The software applies credibility as a second weighting factor for consistency.

Consistency: Consistency ratings occur at the intersection between evidence and a particular hypothesis (see Figure 5.8). Analysts are asked to consider: 'Assuming this

hypothesis is true, what as the likelihood that I would find the given evidence' (i.e., in probability notation they are being asked to estimate $P(e_x|h_y)$, where e_x is 'evidence item x should be found', and h_y is 'hypothesis y'). Possible ratings are: highly consistent (++), consistent (+), neutral (/), inconsistent (-), highly inconsistent (- -), and irrelevant or unrated (n/r). In keeping with the goal of *disproof*, only *inconsistent* or *highly inconsistent* ratings contribute to an increase in the weighted inconsistency score.

Weighted Inconsistency Score: An aggregate score assigned to each hypothesis that sums up the inconsistency of each evidence item (weighted by relevance and credibility), and ultimately forms the basis of comparison between hypotheses (see numbers above the hypothesis columns in Figure 5.8). The values are calculated by multiplying an internal numeric representation of the 'credibility' and 'relevance' ratings assigned globally to evidence (see Section 5.7.4) multiplied by the 'consistency' rating assigned to each evidence/hypothesis pair (see Section 5.7.5). These scores are meant to offer a means of helping interpret the input, but analysts are cautioned not to imbue these scores with more power than their underlying subjective judgements can provide.

5.5 Walkthrough Steps

While ACH Walkthrough can be used for ACH analysis generally, we especially intend for it to be used for a collaborative review, where a small team of analysts work together. In particular, we suggest an approach similar to that suggested by Wharton et al. called the "Cognitive Walkthrough" [78], where a team walks through steps, discussing and executing each step together, each team member contributing from their perspective.

Recall that in our field study, we saw a need for *reflective communication*. We suggest that our walkthrough technique will provide strong support for reflective communication. In particular, when reflecting, analysts should discuss the overall direction of the work, the quality of the work, and the methods they are using to achieve their common goal.

As well as a collaborative review, we suggest that ACH analysis involves some work best done by analysts working independently. For example, this might be most appropriate for searching through documents and identifying evidence, and even for many initial assessments

of credibility, relevance, and consistency with hypotheses. Accordingly, we suggest that the best overall strategy for ACH is to alternate between independent work and collaborative reviews facilitated by ACH Walkthrough. The software allows analysis data to be transfered back and forth with spreadsheets [1] to facilitate this kind of alternation.

The collaborative walkthrough is structured into a series of steps, where each is a step in an ACH analysis, together with discussion relevant to that step. To increase the value of the discussion, we suggest that team members adopt roles. For example, one analyst could play the role of a particular expert or organization, and represent that perspective in the discussion. This facilitates a diversity of perspectives in the discussion, and increases the possibility that critical issues will be identified. Heuer and Pherson [36] discuss the advantages of role-play in intelligence analysis, along with related techniques such as devil's advocacy and "red team" analysis.

In the walkthrough, our multiple device architecture also supports multiple perspectives on the data. As illustrated in Figure 5.3, several devices can be used simultaneously with different views (each view is on a different 'tab' in a traditional tabbed display), and any changes made to the data are instantly synchronized. It would be possible, for example, to have two large touch displays, so that one could be used to consider consistency ratings (explained below), and the other could be used to browse related evidence documents. At the same time, individual analysts could check other tabs on the analysis using tablets or smartphones.

The revised steps we propose for collaborative ACH are based on those described in Heuer and Pherson's book [36], and in the tutorial material for the software for ACH developed by the Palo Alto Research Center (PARC) in collaboration with Heuer [59].

We now describe each of the steps. The basic text for some of the steps is drawn from both Heuer and Pherson [36] and PARC ACH [59] as indicated.

Step 1: Select or create an analysis. "Consider if it is appropriate to conduct an ACH. An ACH can help to analyze hypotheses about what is true or what is likely to happen. ACH is of limited value when being used to evaluate alternative courses of action, i.e., decisions based on goals or personal preferences. Consider the makeup of the group

[1] We drafted a custom CSV data structure that supported multiple record types and relationships between them.

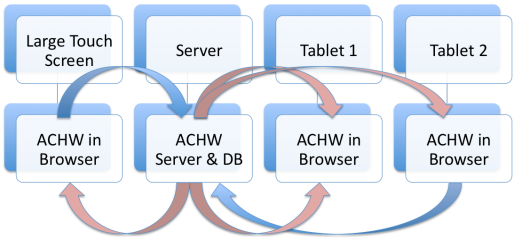


Figure 5.3: Multiple Device Flow: In ACH Walkthrough, any number of devices of various kinds can be used to work on the analysis, and to make changes simultaneously and independently. The changes flow to the server, and thence to any other devices working on the same analysis.

assembled for the walkthrough today. Are there specialists who should be present who are not? Consider soliciting their input individually after this walkthrough or inviting them to the next team walkthrough." [59]

Step 2: Consider each person's role in the walkthrough. Assign one person the role of facilitator. This person will move the team through each of the steps. Assign roles to the rest of the team members. Choose a strategy: 1) Analysts will provide insight based on their particular area of expertise. 2) Analysts will take on a special role such as 'scribe','evidence rating checker', 'devil's advocate', 'mentor', 'the client' 'an intruder', 'another department', 'a particular country' 'a particular person' ... 3) Analysts will be responsible for checking one particular aspect of the analysis such as 'completeness', 'verifiability', 'understandability', ... The selection of roles depends on the state of the analysis so far, and its intended audience. Analysts maintain their specialized role, making comments from their role's perspective as the facilitator walks the group through the steps.

Step 3: Identify possible hypotheses to be considered. "Use a group of analysts with different perspectives to brainstorm the possibilities. Create the hypotheses list." [36] "A hypothesis is a testable proposition about what is true, or about what has, is, or will happen. It should usually be worded as a positive statement that can be disproved. A good set of hypotheses meets two tests. The hypotheses cover all reasonable possibilities, including those that seem unlikely but not impossible. And the hypotheses should be mutually exclusive. That is, if one hypothesis is true, then all other hypotheses must be false. ... When deciding whether to include an unlikely hypothesis, consider

whether the hypothesis is virtually impossible or simply unproven because there is no evidence for it. For example, the possibility that an adversary is trying to deceive you should not be rejected just because you see no evidence of deception." [59]

Step 4: Make a list of evidence. "Make a list of evidence and arguments (including assumptions and logical deductions) for and against each hypothesis." [36] Create the evidence list, set the evidence type. "Assess the Credibility and Relevance of each item of evidence to determine how much weight it should have in the analysis." [36] Use the evidence documents as required. "Evidence is interpreted very broadly. It refers to all factors that influence your judgment about the hypotheses, and this definitely includes assumptions and logical deductions as well as specific items of intelligence reporting. Assumptions or logical deductions about capabilities, intensions [sic], or how things normally work in the foreign country involved are often more important than hard evidence in determining analytical conclusions. Absence of evidence is also evidence and must be noted. For example, if you are analyzing the possibility of military attack by an adversary, the steps the adversary has not taken may be more significant than the observable steps that have been taken." [59]

Step 5: Assess consistency of evidence with hypotheses. Consider each evidence item in turn, and asses the consistency or inconsistency of that item with each hypothesis. "Ask yourself the following question for each hypothesis: If this hypothesis is true, what are all the things that must have happened, or may still be happening, and what evidence of this should I expect to see? Then ask: If I am not seeing this evidence, why not? Is it because it isn't happening, it is being concealed, or because I have not looked for it?" [59]

Step 6: Do a subset analysis. Select and then compare the Inconsistency Scores for different types of evidence, and/or the most diagnostic evidence. Select appropriate subsets of evidence and look at the (Weighted) Inconsistency Scores in the Consistency Tab. Consider other "What-if" scenarios as applicable.

Step 7: Do a sensitivity analysis. "Consider how your conclusion would be affected if key evidence or arguments were wrong, misleading, or subject to a different interpretation. Double check the validity of key diagnostic evidence and arguments that determine

outcome of your analysis. Change the validity ratings of evidence if necessary." [36]

Step 8: Do a sanity check. "Compare your own conclusions about the relative likelihood of the hypotheses with the (Weighted) Inconsistency Scores generated by the software. If they are not similar, figure out why and what you can learn from this." [36]

Step 9: Report conclusions. Make notes for a report, adding to the analysis description. "Include discussion of alternatives that were considered and why they were rejected. Identify milestones for future observation that may indicate events are taking a different course than expected." [36]

5.6 Design Overview

This section presents an overview of the design and implementation of ACH Walkthough. We begin with the User Interaction (UI) design, and then outline the implementation technology that we adopted or developed.

5.6.1 UI Design

ACH Walkthrough is a client-server web application, and it requires login with a userid and password on a project basis. Within a project, the software supports many ACH analyses, each with hypotheses, evidence items, and the scoring of these following the model of Heuer. The UI presents several tabs, where each tab supports one functional aspect of the ACH process. We felt that a tabbed design was consistent with Heuer's step-wise process whereby the user's attention is deliberately tunneled through a structured process.

In addition to the basics of ACH Analysis, the software provides several innovative features to leverage surface computing to support collaboration. These include large-scale touch controls, suitable for small groups, some innovative touch controls we call "fretboards", and a visualization technique called "parallel coordinates" applied to ACH. We also provide "Walkthrough" facilitation to help groups systematically review an ACH analysis. Finally, we use an innovative multi-device approach which allows several devices to be used simultaneously. Each of these is discussed in detail in section 5.7.

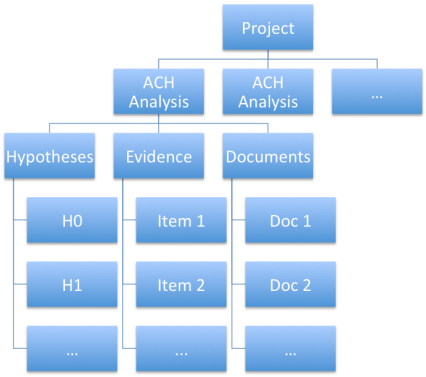


Figure 5.4: Conceptual Structure: Each project may have many analyses, each with several Analyses and many Evidence Items.

5.6.2 Software Implementation

The web-based approach was taken to enable deployment across many platforms with sufficiently powerful and standards-compliant web browsers, and without any need for complex software installation.

As with most web applications, the overall system depends on a central server, with a certain amount of code loaded onto the browser (client) while the software is running. The ACH-W software relies, however, on processing that occurs on both the server and the client. This client application is delivered and updated without interruption or the need for client-side installation. In many cases the server can even be modified and restarted without the client application losing its place.

This approach enables many useful features, such as no data being stored on the client machine when the program is not in use, and the ability for simultaneous use of the software for the same analysis by different devices. The software is written primarily in the JavaScript programming language, using standards compliant language software running in both servers and clients. We use several important external but open-source libraries in our implementation. The implementation is described in more detail in section 5.8.

5.7 UI Design and Rationale

In this section we discuss the UI design of ACH Walkthrough. We begin with a general discussion, and then discuss each of the UI tabs that support the ACH analysis process.

5.7.1 General Concepts

ACH Walkthrough requires authentication, and a userid-password gives access to a *Project*. The conceptual structure used is illustrated in Figure 5.4. Any project may include any number of ACH *Analyses*. Each *Analysis* allows up to 10 *Hypotheses* [2], and any number of *Evidence items*. The program allows creation, deletion, and editing of Analyses, their Hypotheses, and their Evidence Items. It also allows an Analysis to be uploaded from a spreadsheet such as saved by Excel, and also allows an Analysis to downloaded to a spreadsheet.

The main use-case in ACH Walkthrough is performing ACH analysis, as described by Heuer [35]. This consists of identifying Hypothesis and Evidence items, then determining the relevance and credibility of the evidence items, and then their consistency with the hypothesis. In ACH Walkthrough, these steps are supported by separate tabs in the interface, where each tab supports a step in the ACH analysis process. The use of these tabs is shown below.

In addition, ACH Walkthrough has features that leverage surface computing and support collaborative analysis. For example, the primary interface has been designed for use on a large touch-screen for use by a small group engaged in a review of an ACH analysis. The controls, therefore, are large and visible, so use by touch interaction is evident to everyone in the group. These support joint attention and situation awareness. But there are several other features that use touch to support collaboration, as we describe next.

Fretboards: In ACH, there are several steps that involve entry of a quantitative score: credibility and relevance of evidence items, and consistency of evidence with hypotheses. Instead of using numeric entry, we designed a new touch control, the *Fretboard*. The name refers to the fingerboard on a stringed instrument, with lines that mark positions for certain musical notes. Our fretboards allow touch and drag interaction to position an indicator, showing the appropriate quantity. This makes the entry highly visible to the group, and allows spatial reasoning (see Figures 5.7 and 5.8).

Walkthough Advice: In our field study, we identified a need to better facilitate strong collaborative activities such as those involved in joint review. To support this, we

[2] we designed our software so that it could represent up to ten before the display becomes cluttered. In practice there will not be more than a handful of hypotheses, though future editions might consider how best to represent this situation.

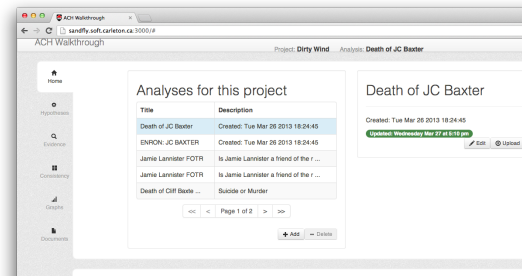


Figure 5.5: Home Tab: Showing list of available Analyses, with title and description, and allowing adding, deleting, and editing, and upload or download to spreadsheet format.

leverage ideas from a kind of software inspection technique called the "Cognitive Walkthrough" [78], hence the name of our tool being ACH Walkthough. The technique involves members of the group selecting roles to play in the review, and then the group stepping through the analysis together discussing each step. This supports a diversity of ideas, and avoids "groupthink". To support this, our tool has "walkthrough notes" that appear and give guidance, as seen in the lower half of Figure 5.8.

Parallel Coordinates Visualization: In our field study and in later exploration of ACH analysis, we found that people wanted to consider the overall patterns in rating evidence for credibility and relevance, and in scoring of hypotheses for consistency. To support this in our tool, we added a visualization of the ACH analysis using the visual formalism known as a "Parallel Coordinates". We considered alternatives [83], but settled on this visualization for its fit to task. Parallel Coordinates is an established visualization technique [39] to aid exploration of diverse data, and the technique has been advocated especially in the context of cyber-security [17]. See Figure 5.9 for an example.

Multiple Devices: One important collaborative characteristic in our software does not involve any specific element in the UI. By leveraging Meteor's automatic synchronization of data across connected clients, multiple screens/users are updated in near real-time, as illustrated in Figure 5.3. This means that, at a meeting, several screens can be used for the same ACH analysis, where changes made on any device are reflected on them all. Multiple large screens may be used, or small tablets. This facilitates parallel work in a collaborative context.

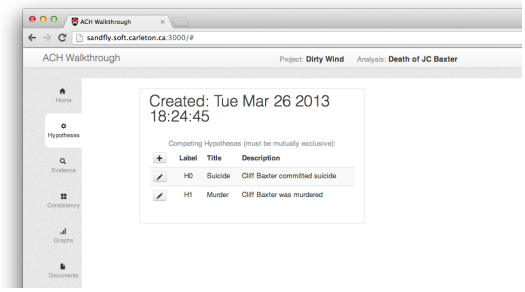


Figure 5.6: Hypothesis Tab: Showing list of identified Hypotheses, with number and description, and allowing creating, deletion, editing.

5.7.2 Home Tab

Usage (see Figure 5.5)

- Lists all available Analyses, showing their title and brief description.
- Touch controls allow analyses to be added, deleted, or modified.
- Selected Analysis can be uploaded from, or downloaded to, a spreadsheet in CSV (comma separated values) format.
- Selection of Analysis populates Hypotheses, Evidence, and Consistency data shown in other tabs.

Design Rationale

- A project can involve many ACH analyses, and this tab allows each to be managed.
- Large controls allow onlookers to easily follow manipulation.
- Upload from spreadsheet format file allows analysis to be done offline in Excel (or other tools), and uploaded for review in ACH Walkthrough.
- Download to spreadsheet format allows analysis to be downloaded from ACH Walkthrough for further offline analysis in Excel (or other tools), or used as a basis for further processing or reporting.

5.7.3 Hypothesis Tab

Usage (see Figure 5.6)

- Lists all Hypotheses, showing their number and description.

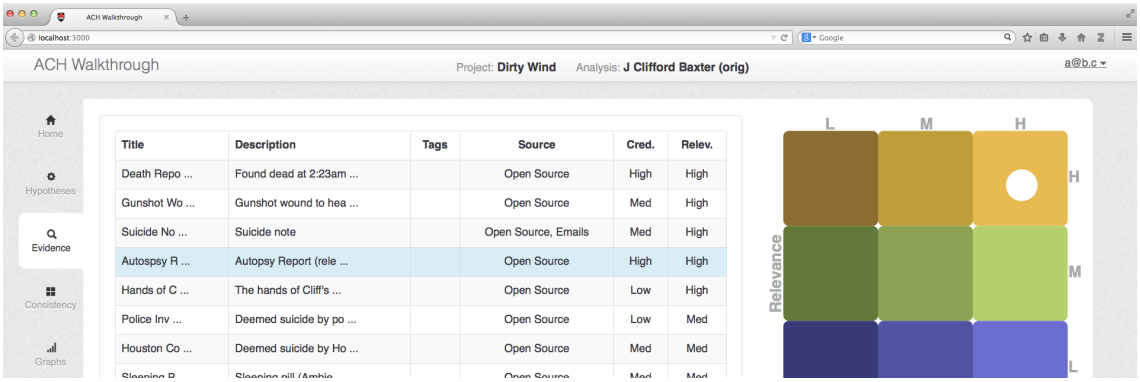


Figure 5.7: Evidence Tab: Showing list of considered evidence with title, description and source. Touch control *fretboard* at right allows setting of credibility and relevance.

- Touch controls allow hypotheses to be added, deleted, or modified.
- Touch of Hypothesis title or description selects it, shows more detail to the right.

Design Rationale

- Large controls allow onlookers to easily follow manipulation.

5.7.4 Evidence Tab

Usage (see Figure 5.7)

- Lists all Evidence Items, showing their title, and description, and source.
- Touch controls allow evidence items to be added or deleted.
- Touch of Hypothesis title or description selects it, allowing setting of credibility and evidence using the Fretboard control at the right, and updating values for credibility and relevance in the list.

Design Rationale

- Fretboard touch device allows both relevance and credibility to be set at the same time in a highly visible way.
- Fretboard allows spatial reasoning about relevance and credibility, and spatial comparison with those for other evidence items.

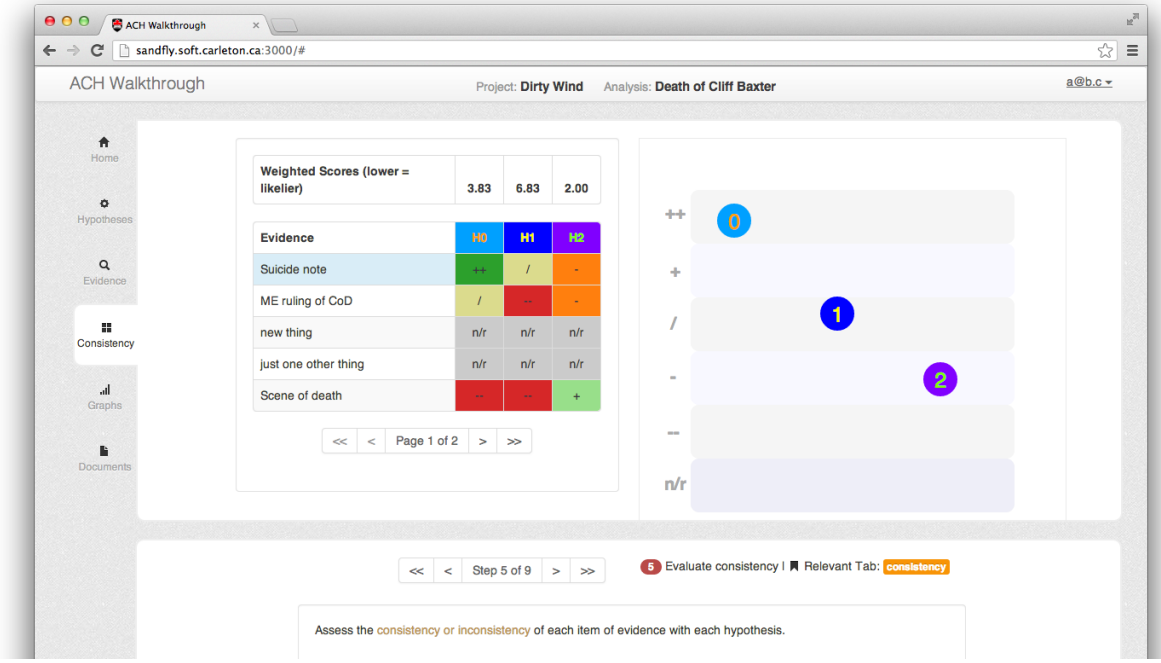


Figure 5.8: Consistency Tab: Showing the main ACH table at left, where the rows show the evidence items, and the columns show the hypotheses. The touch control *fretboard* at right allows setting of consistency ratings. The Walkthrough Advice Panel is shown for step 5 (See section 5.5).

5.7.5 Consistency Tab

Usage (see Figure 5.8)

- Shows an ACH table with a row for each evidence item, and a column for each hypothesis, with the cells showing the consistency ratings for each evidence item with each hypothesis.
- Evidence items may be selected by touch, allowing consistency with each hypothesis to be set using *fretboard* touch control to the right.
- Fretboard control allows consistency to be set, and dynamically updates main ACH table consistency values.

Design Rationale

- Format of main ACH table consistent with Heuer and original Xerox-Parc ACH implementation [35].

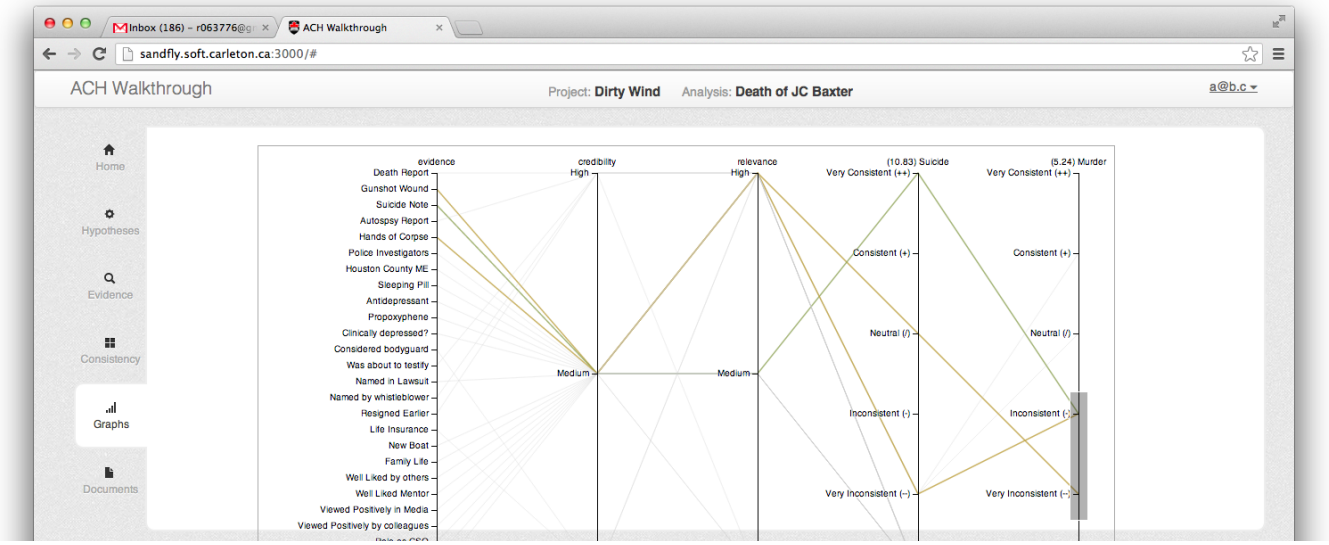


Figure 5.9: Graphs Tab: Showing "Parallel Coordinates" graph visualization for the ACH analysis. Notice the selection box on the rightmost hypothesis axis, restricting selection to the evidence items inconsistent with that hypothesis. This was selected by "brushing" (a term defined by Becker and Cleveland [6] for selection of data within a visualization) that, in this particular case, demonstrates Heuer's goal of seeking highly *diagnostic* evidence.

- Colours on table clearly indicate consistency ratings across evidence and hypotheses.
- *Fretboard* touch control allows consistency settings to be made using spatial reasoning, and with easy comparison between hypotheses.
- *Fretboard* touch controls dynamically alter both main ACH table settings, and their colours, and also overall hypothesis scores along top of table. This facilities "What if?" exploration. For example, what if only a specific subset of the evidence were considered in computing the scores?

5.7.6 Graphs Tab

Usage(see Figure 5.9)

- Shows an Parallel Coordinates graph visualization for the ACH table, where evidence items are shown by lines across, and axes represent credibility, relevance, and consistency scores for each hypothesis.
- Evidence items may be selected by touching on their line, which highlights it, and dims

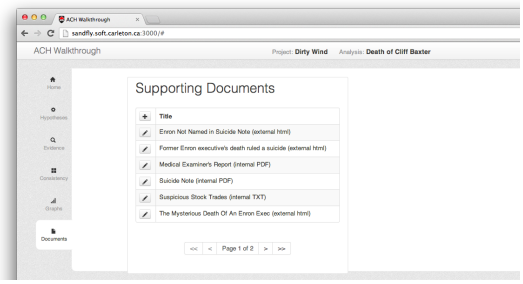


Figure 5.10: Documents Tab: Showing a list of documents related to evidence items.

other lines.

- On each axis, a subrange may be specified by "brushing" a selection box, restricted selection to all lines that cross the axis in that region.

Design Rationale

- Parallel coordinates are an established visualization technique to aid exploration of diverse data, and here intended to facilitate pattern detection and "what if" analysis.
- Selection and subrange selection by brushing allow specification of a subset of the evidence to be inspected.

5.7.7 Documents Tab

Usage (see Figure 5.10)

- Allows evidence documents to be recorded.
- Clicking opens document in a browser tab.

Design Rationale

- Facilitates review of primacy sources.
- Later versions of software could allow highlighting within documents.
- Later versions of software could allow linking evidence items to documents.

5.8 Software Implementation

ACH Walkthrough is implemented as a Meteor application, using the architecture described

in Section 4.3. As a Meteor application it uses JavaScript code on the client and the server. Several external open-source libraries are used, as well as code developed specifically for this application, as discussed below. For presentation in the browser, the HTML5 suite of protocols is used. In particular, format is controlled by CSS (Cascading Style Sheets), and graphical elements are produced using SVG (Scalable Vector Graphics). The use of SVG, rather than the HTML5 Canvas, was selected to facilitate deployment of the application on display devices with diverse screen sizes, from small (such as smartphones) to very large touch-screens.

ACH Walkthrough is intended to work on any modern standard-compliant web browser on any device. Development and primary testing was carried out with server software running on Ubuntu Linux 10.04.4 LTS, and client browser running on Google Chrome Version 25 and Mozilla Firefox Version 19 on both Microsoft Windows 7 and Apple MacOSX 10.8, using the PQ Labs touch frames on large screen LED televisions. Informal testing shows the software also works on a variety of other platforms, including Apple Safari browsers, and Android Operating System tablets. While the technology we use is based on established standards, there is still movement in how touch and gesture interfaces work on different browsers and operating systems. Some adjustment to the software may be necessary for specific platforms and over time.

It should be understood that the current version of ACH Walkthrough is a functional prototype, suitable for demonstration and pilot usage, but not sufficiently complete for institutional deployment.

5.8.1 Frameworks and Libraries

The following libraries represent the core third-party components of our application:

Meteor: Meteor is a JavaScript client-server framework that links data models in client code to a data model on the server. Meteor works in all modern browsers and is open source with the MIT license. For more information see: `https://www.meteor.com`

Blaze: Blaze is the built-in JavaScript template system for Meteor. It allows specification of HTML templates, where dynamic content can be inserted into format structures, and also includes macro-processing facilities to increase flexibility. For more information see: `https://www.meteor.com/blaze`

MongoDB: MongoDB is a database management system oriented around a document object model, facilitating use in web applications. Storage is in BSON (a binary format modelled after the JavaScript Object Notation, JSON), and a variety of query structures are supported. MongoDB is open source with the Free Software Foundation's GNU AGPL v3.0 license. For more information see: `http://mongodb.org`

In addition to Meteor's core components listed above, they also bundle some third party packages. Our software makes use of the following libraries.

D3: D3 (for Data-Driven Documents) is a JavaScript library that connects SVG visual representations to data models. These are coupled so that changes in the model automatically cause changes in the appearance of the visualization. D3 works in all modern browsers and is open source with the BSD license. For more information see: `http://d3js.org/`

jQuery: jQuery is a very general and popular web application library that makes it easy to programmatically manipulate HTML documents using CSS selectors. jQuery is open source with the MIT license. For more information see: `http://jquery.com/`

Backbone: Backbone is a minimalist client-side Model-View-Controller framework we currently only use for its client-side router. Backbone is open source with the MIT license. For more information see: `http://backbonejs.org/`

Bootstrap: Bootstrap is a set of HTML layout components and libraries that Twitter Inc. has shared with the open source software community with the Apache licence. For more information see: `http://twitter.github.com/bootstrap/`

PegJS: PegJS is used as a parser generator for interpreting the CSV import/export for Excel interoperability. Advanced parsing was required because the CSV file contains multiple record formats, and certain child records are related to parent records. PegJS allowed us to describe the import file with a simple grammar, permitting easier re-working of our de-serialization program as the import specification evolved and opening up the possibility for richer formats in future. For more information see: `http://pegjs.org`

5.8.2 Custom Visualizations

Three custom visualizations feature in the implementation of ACH Walkthrough, two *fretboards* and one graph. These custom components were designed to work together to address the need for data input as well as group collaboration as mentioned in the *design rationale* sections of the tabs where they appear. Their features and will be evaluated in more detail in Section 6.2.1.

Fretboards: We use D3 and Meteor to produce a live rendering of two interactive components, the *evidence fretboard* and the *consistency fretboard*. The use of D3 (a.k.a Data-Driven Documents) allows visualizations to be driven from data, but we also turn that around to make it Document-Driven Data. Meteor reflects these data changes locally within the browser and also immediately with any connected users.

Parallel Coordinates: We use also use D3 and Meteor to render our *parallel coordinates* component. Some custom code was required to aggregate updates to multiple collections of data in one visualization. This visualization supports "brushing", which operates as a highly usable query mechanism, enabling a variety of analytical tasks. The ability to retain brush state in session variables enabled independent analysis of subsets of the data.

The Meteor beta release on which we based our project shipped with a toy example for rendering data with D3. This example was used as a starting point for the evidence fretboard. This was reasonably feature complete within a single day. The consistency fretboard required a little more effort. The parallel coordinates graph was orders of magnitude more difficult than the fretboards, requiring the resolution of several problems for which no precedent existed. Of these, the most critical were the following:

1. rule-based rendering (i.e., repaint axes only on certain events)
2. retention of state between tab changes
3. support for brushing
4. support for hover events

Another central challenge, one that we have continued to refine with exploration of subsequent frameworks, is the problem of maintaining interaction state. The brushing feature

of the parallel coordinates graph is a potentially rich source of query expressions, but this sort of retention is typically kept away from the domain model.

In a sensemaking application, queries can be the *essence* of discovery and so they could well become worthy of sharing, and so they must be elevated to the status of first class objects. But still there was a need to retain some separation between *domain* objects and *interaction* objects.

The need for this distinction was resolved for the parallel coordinates display by means of an unpublished client-side data model. In most respects the code for handling such collections is identical to that required for published collections. This feature of Meteor could eventually make it easy to share elements of the interaction model with collaborators on other screens, and it could also be used to replay the unfolding of an analytical process. We will see more about this idea in the next chapter.

Chapter 6

Evaluation

6.1 Context

For our evaluation of ACH Walkthrough we had access to two vital resources. The first resource consisted of senior members of the group from the field study ('the client'), and the second resource was a panel of professors at an American university ('the demo panel') for whom the client requested we give an extended hands-on demo. Both groups provided extensive and helpful feedback which we present in the following sections, organized by major theme.

Evaluation spanned three different occasions. The first event was a product demo session at the client's site, where we were met by most of the field study participants, and they had a chance to offer critical feedback. In a much longer interaction, two of these participants attended our lab for a half-day usability session which generated several of the comments below. These analysts used the tool and followed the steps in conducting an ACH analysis, providing 'think aloud' feedback as they went. The third session was a half-day mock analysis conducted by the demo panel (divided into several teams), where users spent several hours in hands-on use with the tool, and where I visited each team in turn and discussed their experiences. Roughly half of these teams also provided written feedback after the session.

6.2 Usability of Interactive Visualizations

We begin by discussing the visualizations. These features were central to our interaction design, and they produced the most reviews, both positive and negative.

In this section we will draw from Ware's research [77] and explore arguments for and against two of the most popular forms of multi-dimensional data visualization, namely the *parallel coordinates graph* [38, 77] and the *generalized draftsman's plot* [77] (for examples of these graphs see Figure 6.1 b and a respectively). While both leverage the natural human

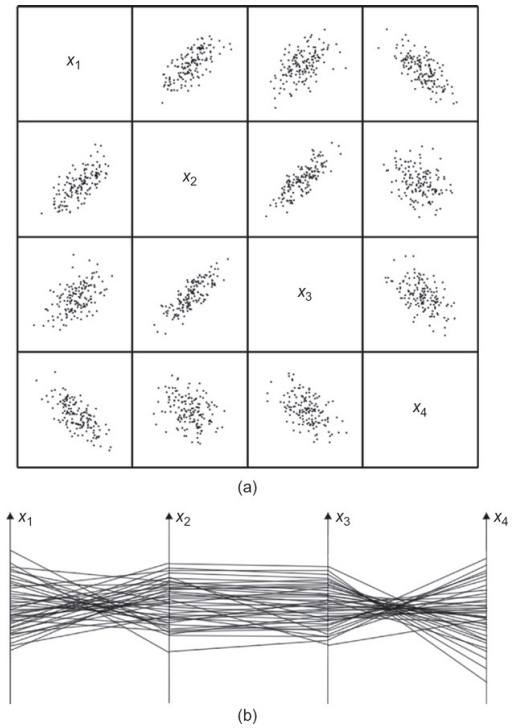


Figure 6.1: Multi-dimensional Visualization Examples. Source: Li et al. [47]

ability to perceive patterns, they each have strengths and weaknesses. We will show why the parallel coordinates graph was the better choice for the particular set of tasks emerging in conducting an ACH.

6.2.1 Design Rationale for Parallel Coordinates

The *parallel coordinates graph* (hereafter *par-coords*, shown in Figure 6.1b) displays all dimensions in a single visualization. Each vertical axis represents one dimension, and data are plotted by lines joining adjacent axes. Ware notes that the resulting output depends on the order of axes. For example, in Figure 6.1b there is no direct relationship represented between dimensions x_1 and x_3.

While the perceptual properties of these visualizations are interesting, their power as tools for exploration can be greatly enhanced with live interaction. For example, the missing comparisons in the *par-coords* graph could be easily seen if the axes could be manually re-ordered.

A second common interaction technique called *brushing* enables the user to highlight a

subset of the data points. This selection typically extends across all other dimensions and so may even provide clues to relationships beyond simple pairs. This works well in both the *par-coords* and the generalized draftsman plot, reducing visual clutter and allowing users to highlight regions with task-specific meaning.

For one such task, recall that Heuer's ACH focuses on *disproof*, and he would argue that 26 years of its continued use in at the CIA in Deception Analysis [34] might demonstrate that *disproof* is his technique's greatest strength. His goal was to make analysts more skeptical of their favourite theories. *Brushing* the 'inconsistent' range of one's favourite hypothesis will immediately highlight evidence that tends to refute it (brushing is explained in more detail below). This is only one of several tasks that have an important persuasive effect, and we chose a parallel coordinates representation for its power to support these tasks quite convincingly. Additional confidence in the conclusions comes from the structured evaluation of evidence as it relates to all possible theories.

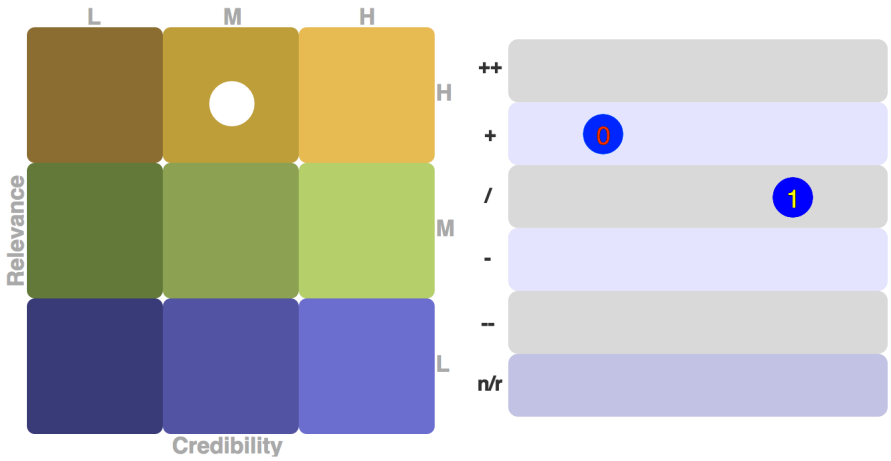
For ACH Walkthrough, three custom visualizations were developed to address the needs of co-located and remote collaboration, and these visualizations were used for the following tasks:

1. When entering evidence, users rank credibility and relevance on a 3x3 matrix (Figure 6.2a)
2. When evaluating diagnosticity, users rank consistency using a *fretboard* (figures 6.2b and 6.2c)
3. When refining the matrix, users employed an interactive parallel-coordinates graph (Figure 6.3)

The *par-coords* axes come from the following dimensions of the application database:

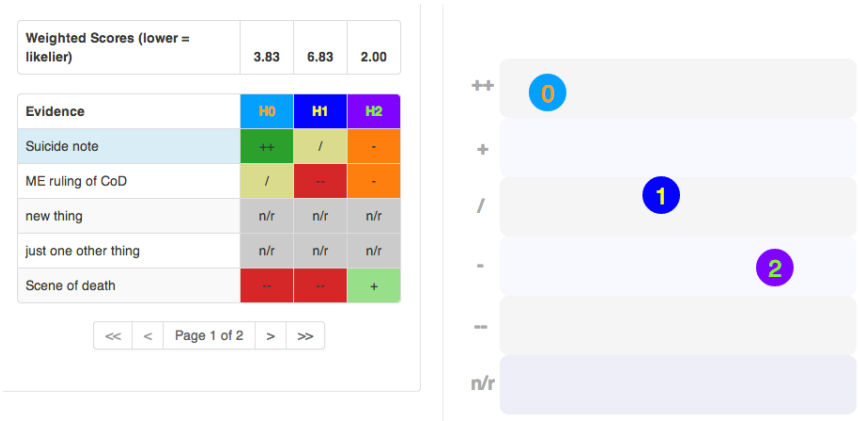
1. Evidence name from the evidence table
2. Credibility from the Evidence rating widget
3. Relevance from the Evidence rating widget
4. One axis for each hypothesis from the Diagnosticity widget

Without repeating all the details of ACH, a few factors of the resulting data should be mentioned. The evidence ratings of relevance and credibility are saved on an *ordinal scale*



(a) 3x3 Evidence Rating Widget

(b) Consistency Widget



(c) Widget in context

Figure 6.2: ACH-W Input Widgets

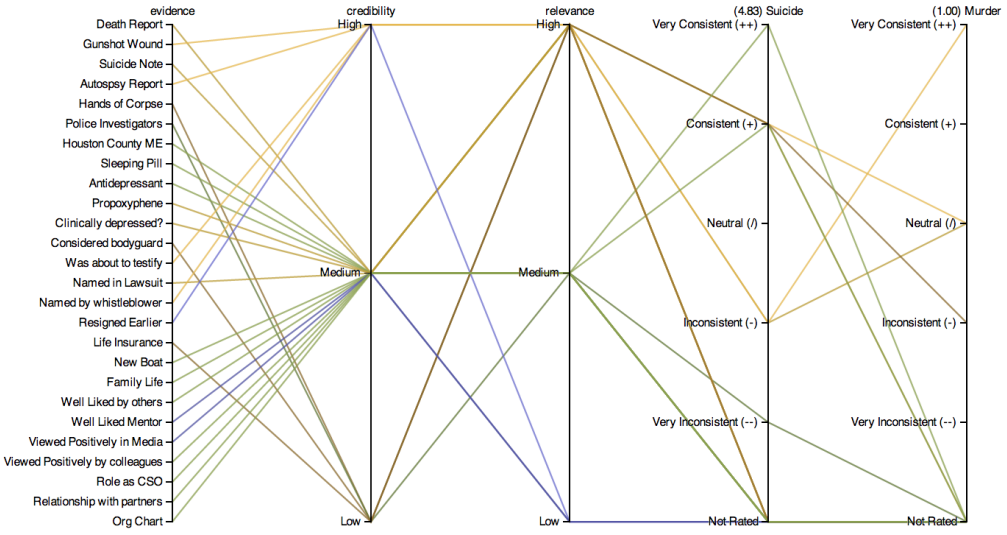
of Low, Medium, and High. The consistency ratings – answers to the question: *"Is evidence item E consistent or inconsistent with hypothesis H?"* – are similarly rated on an *ordinal scale* of: highly consistent (++), consistent (+), neutral (/), inconsistent (-), highly inconsistent (--), and irrelevant or unrated (n/r). The fact that these are ordinal values and not *ratio data* come originally from the Xerox PARC implementation mentioned earlier, and we could have used our input widgets to store more precise *ratio data* (in fact we do store ratio data, but we convert it to ordinal data for calculation and display). As it turns out there is an important visual benefit to these scales being non-continuous which we will discuss in Section 6.2.4.

Figure 6.3a shows the first version of a *par-coords* plot for ACH-W, and an important issue should be immediately apparent. The problem can be seen when examining the number of lines between the first and second axes (left to right) and the apparent loss of detail as lines in subsequent gaps overlap. This problem results from the fact that the data points are not floating point values but instead are categorical (the first axis) and ordinal (the remaining axes). This loss of information can be corrected by using curved lines, as shown in figures 6.3b and 6.3c.

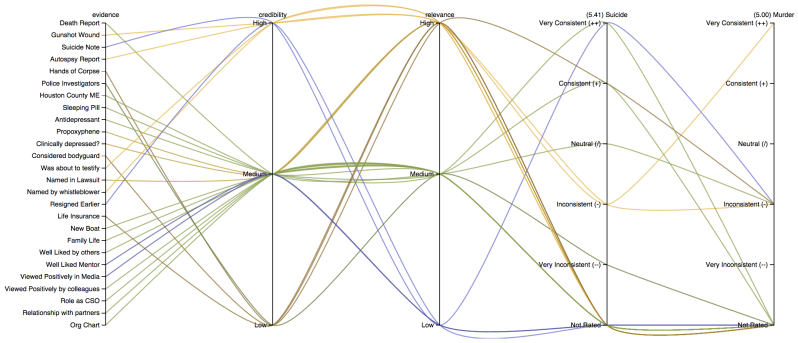
With this problem corrected, we can begin to see the value of colour in the overall process in ACH-W (we assume you are reading this in colour). The colours from the *Evidence Rating Widget* carry over into the *par-coords* graph. This can be seen most clearly in Figure 6.3c. Notice the shades of orange, green, and blue. Users familiar with this coding scheme will recognize this as relating to relevance and credibility, for example bluer lines are less relevant to the overall question than lines rendered in greens and oranges, and brighter hues represent greater credibility.

We note in passing that one of our reviewers suffered from colour blindness, and while this did not detract from his ability to use the system he did miss some of these subtle cues. He requested future versions enable selection of colour ranges for the widgets, reflected of course in the parallel coordinates line colours.

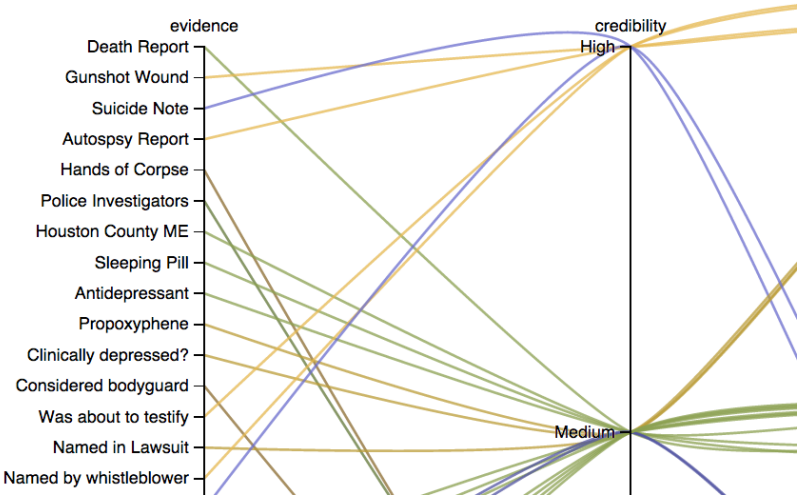
Our use of colour offers users an overall interpretive cue for analysis, but the real power of the tool comes from interacting with the graph's axes. Selecting sub-ranges of the axes using *brushing* allows the user to focus on meaningful subsets of a dimension's overall range. Lines that fall within these highlighted regions retain their brighter colours, while lines that fall outside the regions are rendered in muted grey. This operates as a filter query that permits



(a) Original version



(b) Correcting for non-floating data



(c) Enlarged view showing improvement with curves

Figure 6.3: ACH-W Parallel Coordinates

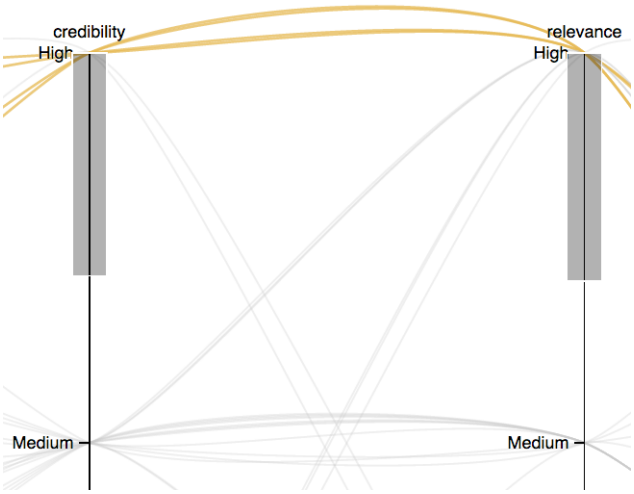


Figure 6.4: Selecting for high credibility and relevance

easier inspection of features of interest.

The brushed ranges are retained until deliberately removed. This allows the user to add additional brushes on other dimensions, effectively narrowing their query with a great deal of flexibility. For example, Figure 6.4 is a spotlighted section from a larger graph. It shows the effect when seeking only highly credible and highly relevant evidence.

Brushing supports a surprising range of interaction tasks, especially as users become familiar with the meaning of the graph's dimensions. Users new to *par-coords* graphs might at first be drawn to visual clusters and reinforcing trends across the display, and indeed in many domains this is a strength of *par-coords* in general. In the particular case of analysis work like ACH, the real power comes from drawing one's attention to *individual* evidence items that fall within meaningful regions of the graph and then taking the time to consider one's evaluations from fresh perspectives.

Heuer's technique of ACH was designed to draw attention to highly *diagnostic* evidence when the natural human tendency is to explore evidence that confirms our suspicions. He explains diagnosticity with an example taken from medicine. A junior medical intern recently swamped by flu season might incorrectly diagnose flu again based on yet another patient complaining of fever, whereas a more experienced resident would know that fever is consistent with too many conditions for it to offer much diagnostic utility and they would ask more probing questions. Their focus would be on identifying symptoms that have the power to *rule out* competing hypotheses, rather than those that confirm the diagnosis they most easily

suspect.

Interaction with *par-coords* supports this kind of diagnostic reasoning by making it easy to select items that help rule out a given hypothesis. The user can create a brush that selects for ratings of *inconsistent* or *very inconsistent* along a particular hypothesis axis and then draw their attention to the evidence associated with only the highlighted lines. If they had previously defined brushes for high credibility and relevance (Figure 6.4), they would quickly find evidence items requiring the greatest consideration.

6.2.2 Parallel Coordinates Affordances for Touch

Our informal usability sessions revealed opportunities for refinements of interactive features. Our internal testing using brushing and parallel coordinates had shown it offered powerful analytic value, but in user testing we learned that it does depend on some prior awareness of the brushing as well as a certain level of patience and attention to detail. We had assumed that most users would have encountered interactive displays in web forms, but for several users (particularly those new to parallel coordinates), the availability of brushing was not immediately obvious. It may have gone against their expectations if they assumed that the visualization was merely a static aggregation of data.

Without cues from experienced users, our testers did not attempt to apply any brushes. In our current implementation of ACH-W there aren't any obvious interaction cues (a.k.a. affordances) for a newcomers. In fact there is only one type of discoverable affordance and it is offered to mouse users when hovering the pointer over an axis. Unfortunately this feature assumed that hovering could even take place. Users of touch interfaces lack the ability to hover, and so they miss out on interaction cues altogether.

This issue became apparent through a usability test where the participant was helpfully thinking aloud and found himself stuck on one of the walkthrough steps. It was only the novelty of the technique that caused a problem. Once he was shown the availability of the brush feature, its meaning was readily apparent.

Even when users understood brushing, they did not immediately grasp its ability to help seek evidence that could *disprove* their favourite hypothesis, a task that is central to reducing cognitive bias. One possible enhancement for first-time users might be to introduce the feature of brush-based filtering by offering a list of pre-set selections based on ACH-specific

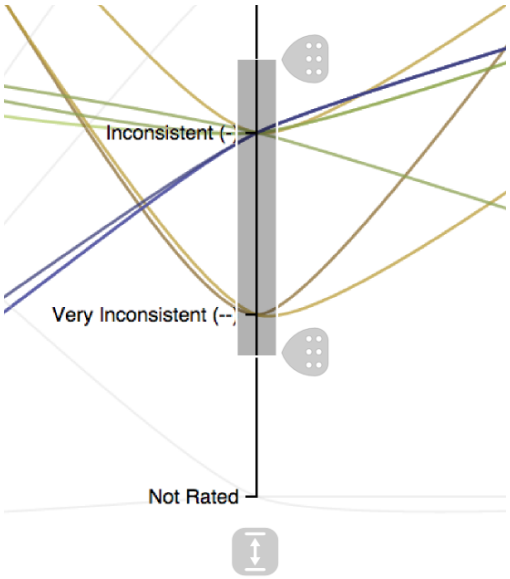


Figure 6.5: Mockup of possible affordances.

tasks (e.g. filter irrelevant items, confirm diagnostic items for hypothesis n, then $n+1$, find counter-evidence for hypothesis n, etc.) and then instruct the user to walk through each of these presets. Also, the initial rendering of the *par-coords* graph could briefly show animated selections on each axis that quickly unfold until they encompass their full range and then leave behind affordances for the user to adjust (see mockup in Figure 6.5).

6.2.3 Implied Logic with Multiple Brushes

Another source of confusion revealed in usability sessions was that multiple brush filters operate as an implied logical AND condition. In other words, when more than one axis had a sub-range selected, items that that did not satisfy *all* filters were suppressed. More than one user had assumed a logical OR condition would apply, and they expressed surprised at some of the results when multiple filters were active.

A somewhat advanced solution would be give the user control over the implied query. For example we could allow users to save and name individual brush selections (filters) and then combine them using a separate expression editor. This editor could be a graphical formalism that could employ a tree structure for nested logical expressions, perhaps accompanied by a syntactical form of the entire expression in a text editor.

A mockup logic flow diagram for an advanced query solution is shown in Figure 6.6. The

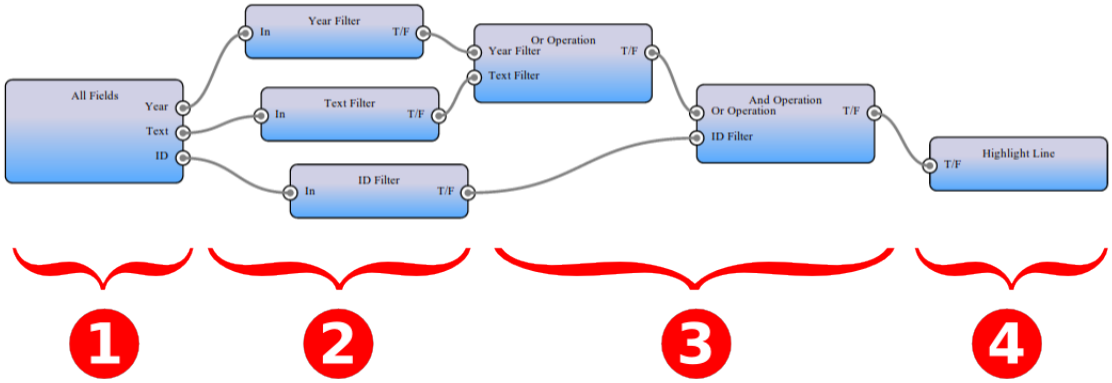


Figure 6.6: Mockup of brushing logic editor.

diagram was produced using JavaScript code from a CodeProject article [19], demonstrating that it could be blended with code already produced for our case studies. The design concept is that the leftmost element (1) would contain outputs for each axis brush on the *par-coords* diagram. Note that only a single brushed segment per axis is currently supported, but with this more expressive editor it is possible to define more than one brush on a given axis. These brush outputs would pass through configurable filters (2), and the output of these filters would be joined through AND or OR operators (3). The final results would be piped to a rendering operation (4) that could be used to configure display options such as line thickness, curve smoothing operators, and hover properties. The logic produced by the editor could also be displayed in a text-based boolean expression, and this expression could be saved for future use.

6.2.4 Representing Data Dimensions with Colour

In this section we explain our decision to use ordinal scales for the colour dimensions in the parallel coordinates graph. This addresses one reviewers' insightful question: Since our ratings of all these values subjective, could you have used continuous colour scales on the Evidence Rating Widget and the Consistency Widget?

As mentioned earlier, we divide our input widgets into sections that map continuous values into discrete ordinal values. This corresponds with the Xerox PARC implementation, and it ultimately lets us use their calculation for overall hypothesis scores. But there is another reason why categorical data is of value, and that is that there are human factors that come into play that argue against using continuous colour scales.

The problems relate to our limited ability to make subtle distinctions with certain stimuli. This is best articulated in Weber's Law of *just noticeable difference* and Stevens' *Power Law* (see Chaudhuri [13] chapter 1). They explain that each type of sensory stimulus has more or less capability to evoke fine distinctions, and fine distinctions along a colour spectrum is one of the poorer types of stimuli. Further experiments with colour have shown that those fine distinctions aren't even linear, making it diffucult to judge, for instance if one grey is twice as dark as another. In addition to these *perceptual* limits there are also *semantic* ones. Even if it were possible to perceive subtle distinctions between two colours, most people can't put names to more than a few [77]. That makes colour excellent for a few categories, but *only* for a few.

In our parallel coordinates display we had already represented the ratings for relevance and credibility along spatial axes, and these values were simply repeated in colour as a means to help trace lines across hypothesis axes. On the strength of the problems of perception and semantic distinction, we felt that we should continue to use the original ordinal scales defined in Xerox PARC's implementation.

6.3 Alternative to Parallel Coordinates

This section addresses the question: Could we have used a different visualization to represent the analysis?

We did explore alternative visualizations. Given our goal of understanding the relationships between dimensions, one simple solution would have been to render all possible pair-wise comparisons. The *generalized draftsman's plot* (GDP) does precisely this. In its simplest form we find a matrix of scatter plots (Figure 6.1a). While this does address the lost detail mentioned above, Ware points out that what it gains in detail it loses in ease of simultaneous comparisons between two or more dimensions.

We mocked up a more advanced form for comparison (see Figure 6.7), where we took advantage of the redundancy across the diagonal to make space for additional controls.

GDPs are perfect for pairwise comparisons, but we had to rule them out for our use case. First, it was difficult to do more than pairwise comparisons. There are cases where the user must consider multiple dimensions at once, something that brushing on a parallel coordinates graph. This wasn't possible with a GDP. Secondly, it was essential to trace the data points

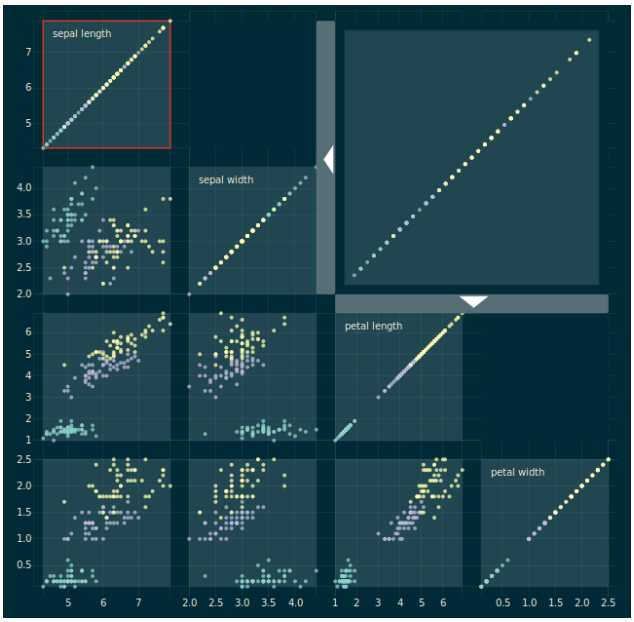


Figure 6.7: D-plot with 4 dimensions

back to the related evidence text, and the GDP could not support this requirement. Finally, the data from our system was categorical and not continuous, and this would have caused serious occlusion problems as multiple data points landed on the exact same positions. This could have been adjusted by binning the values and representing a larger data point, or by adding a random offset to counteract the occlusion, but given the fact that there were already serious traceability problems, this option was set aside.

6.4 Critique of the ACH Process Itself

A number of senior researchers from the demo panel expressed concerns with the process of ACH in its present form. Their concerns fell into two broad categories: psychological and mathematical. We will include here representative arguments that offer the possibility of improvement using changes to the software.

One concern was that the insistence in ACH of mutually exclusive hypotheses might lead to black-and-white thinking (or the *fallacy of the excluded middle*), and there were recommended solutions on two fronts: 1) use of search tools relating to a corpus of reference topics to assist in brainstorming hypotheses, 2) a possible research project on the use of Natural

Language Processing tools and Machine Learning to suggest/review possible hypothesis generating questions, if not the actual hypotheses themselves. The first of these tools might well be within the scope of future enhancement, and the second sounds like an excellent subject for a PhD topic. Unfortunately both were outside the scope this Master's thesis.

A second concern related to our use of the formula for consistency scoring originally produced by Xerox PARC and reproduced in other tools, a formula which we adopted to be congruent with currently deployed tools available for ACH. This formula was contested by the mathematicians from the demo panel on the grounds that there was no formal theory to support it. We proposed that future versions of ACH-W could offer a plugin mechanism for converting user ratings into consistency scores. It could also show different scores based on user-selectable setting, provided the user was also made aware of the theoretical basis for each calculation.

6.5 The Software Features of ACH-W

Another concern was much easier to consider addressing. The commenter was concerned that the two digits of precision used in presenting scores against hypotheses in the Consistency Tab and the Graphs Tab might mislead the user into perceiving a mathematical distinction between closely ranked scores. The scores use a formula developed with Heuer in the construction of the Xerox PARC version of ACH, and we chose to reproduce this formula. Future versions will represent a more coarse representation of score, or may eliminate the numeric score entirely and instead use a visualization that fosters appropriate attention to the similarity rather than the minor differences between hypotheses.

Similar to this concern was a comment on the immediate feedback of the change in score provided while manipulating the ratings on the Consistency tab. The reviewer believed the immediate feedback might actually encourage confirmation bias rather than fight it. This was an interesting concern that could form the basis of a future experimental review. Design of such an experiment could prove difficult to achieve however, particularly given the various other natural sources of confirmation bias present. It would also be difficult to produce a baseline from which to establish the presence of an effect. This was left as another potential avenue for future research.

6.5.1 Top Requested Features

The following features were requested by one or more reviewers:

1. Versioning and merging of versions
2. Roll-back and play-back functions
3. Improved support for integration with external data sources
4. Bidirectional links between evidence and precisely tagged supporting documentation
5. Voice input for hypotheses and evidence
6. Colour customization for rating system (from a colour-blind evaluator)

6.6 Changes Undertaken

The reviews of ACH-W led to a number of prototypes, two of which are becoming interesting projects on their own.

6.6.1 Lowering Friction

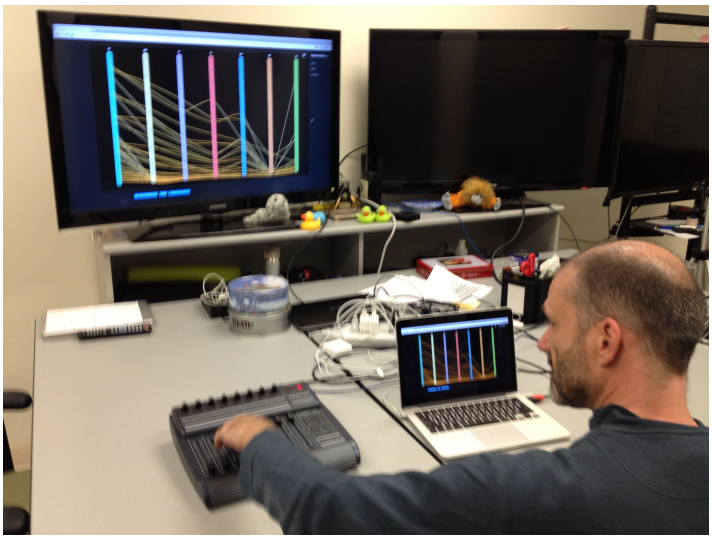
The call for voice input (feature 5 from Section 6.5.1) raised an interesting question: If evidence can be verbally tagged, and if people like the parallel coordinates view (they report that they do), then just how quickly could we get information into it? We mocked up some quick input methods for use with this type of graph. The result is shown in Figure 6.8.

We built a prototype for controlling a parallel coordinates view's brushes with a MIDI device (see Figure 6.8a). The device is motorized, making it possible to control via the touch-screen as well. For rapid data entry we used a Leap Motion detector (figures 6.8b and 6.8c) to detect gestures in mid-air. A new record could be quickly set and evaluated, while voice input described the evidence. When the entry was done, a "thumbs-up" gesture could save it to the database.

This work is still in process, and definitely would require experimental validation to show the quick input did not work against the methodical analytical approach defined by Heuer.

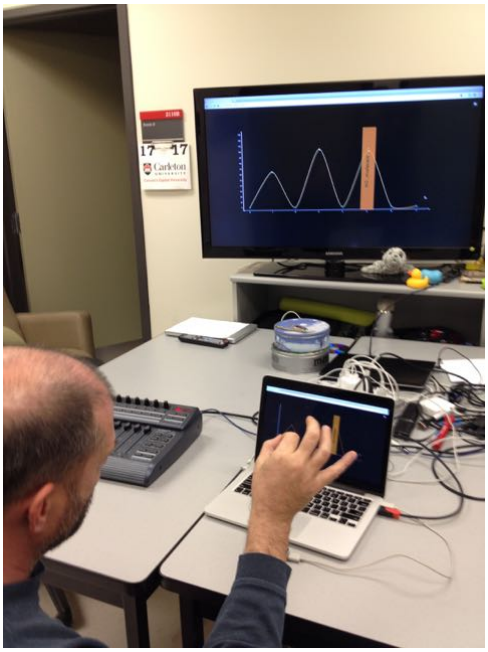
6.6.2 Supporting History

Initial efforts to support history and versioning (features 1 and 2 from Section 6.5.1) were initiated by another student in our lab, Peter Simonyi, whose work on a history plug-in [68]



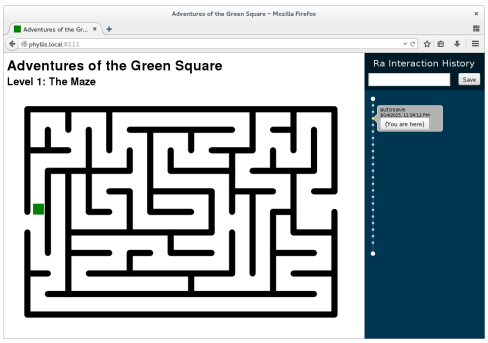
(a) MIDI Input

(b) Choosing an axis using Leap Motion by pointing in space

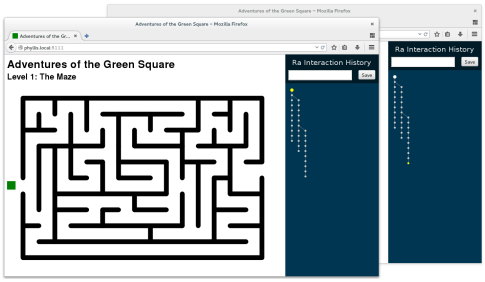


(c) Setting a value using Leap Motion by pinching and dragging in space

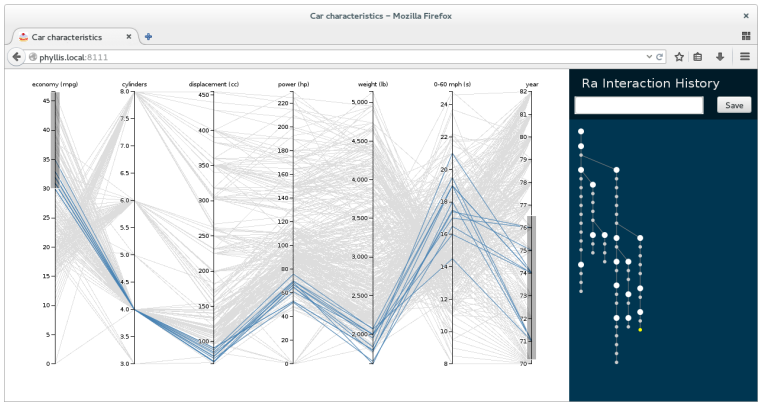
Figure 6.8: Quick Entry Prototypes



(a) Choosing from prior history (you are here)



(b) History is saved centrally for collaboration



(c) Plugin architecture allows different applications

Figure 6.9: Working with History in RA

will be an important addition to future versions. On some of our longer analyses the accrued effort gradually appeared to evoke an unwillingness to risk losing an important mental thread. This is a problem that must be overcome if analysts are going to be willing to question their own biases and conduct appropriate review phases. The provision in future versions of lightweight history tools could reduce the fear of losing context for their thoughts and foster the kind of confidence required for exploring the "what-if" questions that offer the best hope for overcoming cognitive bias.

The plugin (named *Ra*) uses features of the latest edition of JavaScript called Proxy objects, to enable web UI programmers to save changes to history. This process happens automatically on a configurable frequency or under the user's control from a "Save" button in the history panel. Figure 6.9 shows two different applications with the RA plugin enabled. The rightmost panel in each image contains the interaction history as it unfolds. Users are able to choose from any point in the history and proceed from there (see Figure 6.9a), and

unlike the standard undo in browsers and editors the older branches in the tree can still be reached. We believe these two features, namely the *visibility* of a version tree coupled with the *perservation* of all versions regardless of outcome are essential to instilling confidence in "what-if" exploration.

6.6.3 Supporting Annotation

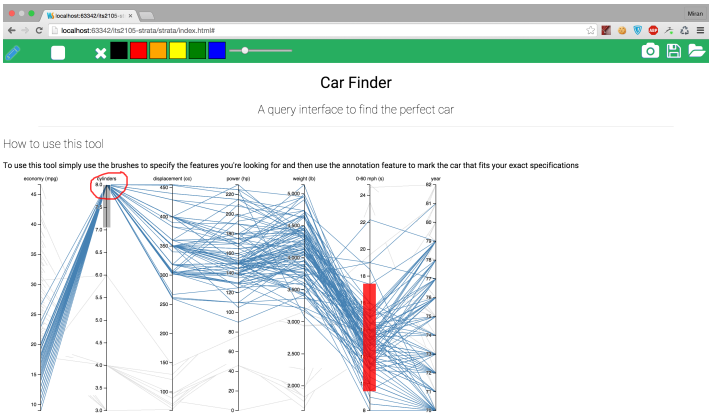
The need to improve context underpins another spin-off project in our lab, a plugin called Strata [52] for the rapid sketching of annotations, developed this summer by co-op student Marin Mirza. This plugin goes partway to improving connections with material in supporting documentation (features 3 and 4 from Section 6.5.1), and it also provides an excellent general tool for collaboration and traceability.

We initially conceived of this feature as helping with multi-device and remote collaboration, enabling users to point to interesting aspects of an analysis underway. This idea is supported by the freehand annotation feature in Strata Figure 6.10a, but Miran has taken the idea farther, supporting highlighting with semi-opaque circles and rectangles. This feature could potentially be used in conjunction with an Evidence browser provided the annotations were able to match up to scrolled documents. Given the need for rich documents from potentially proprietary sources, the full requirements for this feature are being considered for future work.

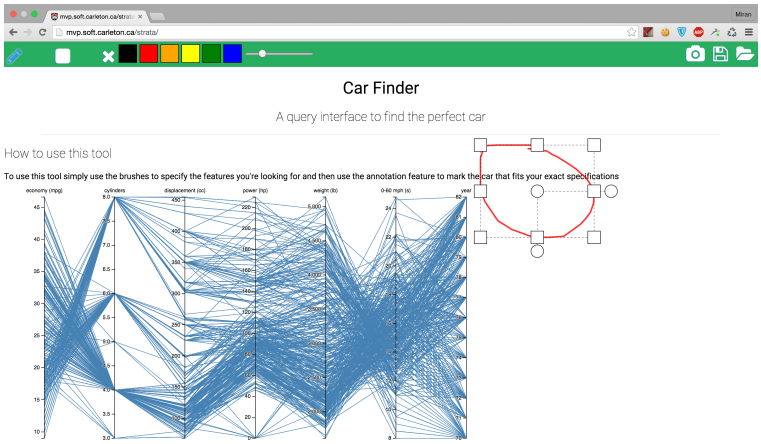
6.7 Software Architecture

We now turn to a brief discussion of the software architecture of ACH-W. Some maintenance on ACH-W was performed by a second developer some time after its evaluation. This late arrival to the project had some surprising challenges modifying the software, a problem that raised concerns regarding the the system's maintainability. Meteor was still new when we adopted it, and the community around it had not yet found ways to break free of its more opinionated architectural decisions. For some developers these decisions represent unreasonable constraints, and so after the first presentations of the ACH-W prototype we took yet another step back from our implementation to look once more at more abstract architectural building blocks.

In Section 6.5.1 we referred to two projects that were spun off after the first review cycles



(a) Annotations can be free-form or shapes



(b) Annotations can be moved, resized, and rotated

Figure 6.10: Working with Annotations in Strata

of the ACH-W project. The requirements of these features played a part in articulating any future architectural requirements.

It must be mentioned that ACH-W was developed at perhaps the most turbulent transitional periods in the history of web development. These few years have witnessed an explosion of innovation surrounding web application design. When it comes to support for multitouch on the desktop — an essential feature for large surface deployments — there is still no cross-browser solution, and as of August 2015 Firefox has disabled support for multitouch on their desktop browser [32].

The result of our discussions, coupled with the discovery of some performance issues as certain analyses became more complex, suggested that the software might benefit from some of the new approaches to DOM manipulation not available at the time ACH-W was initially written.

Appendix A is a collection of writings that began to address the need for these improvements in ongoing projects with our government partner. The first post introduces the approach taken while building the *low friction* prototype discussed in Section 6.6.1. The second post introduces some of the difficulty in working with state management, particularly in the context of libraries that also manipulate state. This is particularly important when handling state in undo/redo scenarios, or the more complete history tools like the Ra Project described in Section 6.6.2.

Chapter 7

Conclusions

7.1 Overview

The intent of this thesis was to describe the process of building a prototype web application for large surface displays that supports collaborative sensemaking using a technique called Analysis of Competing Hypotheses.

Chapter 1 introduced the topic of sensemaking and then we laid out the plan for the rest of the thesis. In Chapter 2 we summarized three important papers from the literature on sensemaking, outlining its iterative nature, its use of flexible conceptual schemas and semantically meaningful collections, and the value large surfaces can offer in supporting the exercise. Chapter 3 described our study of a number of application development practices and frameworks that were focused on large multitouch displays. In Chapter 4 we described our efforts to produce a web architecture that would support a reasonable subset of desirable features from Chapter 3. In Chapter 5 we summarized the field study that motivated our choice of analytical techique (namely ACH), we outlined the requirements that emerged from the study, and then we proceeded to describe in detail the features of our prototype. In Chapter 6 we described the reaction to our prototype, and then we delved in some detail into avenues for future improvements and research.

7.2 Contributions

The primary contributions in this thesis were:

1. ACH Walkthrough, a web-based application for large displays designed to support intelligence analysts in the effort of sensemaking.
2. The introduction of two new interactive visualizations in ACH-W (the evidence widget and the consistency fretboard) that combine the direct manipulation of input data with

important visual cues to collaborators. In addition, we made innovative use of parallel coordinates visualization to assist with the later stages of the analysis.

3. An exploration of web development architectures for large surface, teamroom applications of this type. In particular we demonstrated the value of a data-driven architecture of the sort provided by Meteor, and how well this integrated with data consuming visualization libraries like D3.

We believe the two most important design principles emerging from our development and use of the application were these: 1) make every effort to enable *low friction input*, and 2) throughout the process foster trust that the system supports *fearless exploration*. Incorporating the two spin-off projects already launched by our lab collaborators are the first important steps in this efforts.

We note that the rapidly evolving nature of web application design may limit the useful lifetime of our chosen architecture, but we trust that our work capturing the essential features of designing large multitouch interfaces will hold its value.

We also recognize the limitation imposed by the familiar UI structure of tabs and lists, and we note that this may have reduced the power offered by large displays as articulated by Andrews et al. [4].

7.3 Future Work

We have begun to examine the *low friction* principle with the use of 3D input tools like Leap Motion for rapid data entry and manipulation of parallel coordinates. In answer to the need for *fearless exploration* we will be integrating the history plugin as well in addition to exploring the more detailed semantics of user queries arising from parallel coordinates manipulation.

We also see value of future work to experimentally validate the cognitive benefits of the design principles articulated here. We believe such validation is definitely achievable and it is probably worth doing. Success in the incremental benefits of software improvements here may well provide an avenue for experimentally validating the ability of ACH to reduce cognitive bias, a conjecture currently only supported by anecdotal evidence under conditions that would be impossible to replicate in the lab.

We believe that if ACH could be experimentally validated it could be used in a wide variety of applications. It could be used to help make important life decisions, such as the choice of academic major or institution. It could also prove to be a valuable with broader social consequences, such as in helping with parole decisions or in understanding an unanticipated market failure. Proving experimentally *that* it works is essential, and we believe it might be possible by proving *exactly how* applications like ACH-W might make a difference.

Bibliography

[1] Meteor. http://meteor.com/.

[2] Custom elements: W3c editor's draft. http://w3c.github.io/webcomponents/spec/custom/, 2015.

[3] Anand Agarawala and Ravin Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1283–1292, New York, NY, USA, 2006. ACM.

[4] Christopher Andrews, Alex Endert, and Chris North. Space to think: large high-resolution displays for sensemaking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 55–64. ACM, 2010.

[5] Dmitry Baranovskiy. Raphaël. http://raphaeljs.com/.

[6] Richard A Becker and William S Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[7] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 163–172, New York, NY, USA, 2005. ACM.

[8] Michael Bostock. D3.js. Available at: http://d3js.org/.

[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[10] Judith M. Brown, Jeff Wilson, and Robert Biddle. A study of an intelligence analysis team and their collaborative artifacts. *School of Computer Science Technical Report TR-14-04 , Carleton Uni.*, 2014.

[11] Judith M. Brown, Jeff Wilson, Stevenson Gossage, Chris Hack, and Robert Biddle. *Surface Computing and Collaborative Analysis Work.* Synthesis Lectures on Human-Centered Informatics. Morgan & Claypool, 2013.

[12] M. Burton. ACH:A free, open source tool for complex research problems. http://competinghypotheses.org/. Accessed: 2014-09-03.

[13] A. Chaudhuri. *Fundamentals of Sensory Perception.* Oxford University Press, Incorporated, 2011.

[14] Herbert H Clark and Susan E Brennan. Chapter 7 grounding in communication. *Bach*, 13(1991):127–149, 1982.

[15] UC Berkeley CNMAT. Open sound control. `http://opensoundcontrol.org/introduction-osc`.

[16] Alistair Cockburn. *Agile Software Development: The Cooperative Game (2nd Edition) (Agile Software Development Series)*. Addison-Wesley Professional, 2006.

[17] Greg Conti. *Security data visualization: Graphical techniques for network analysis*. No Starch Press, 2007.

[18] Multiple Contributors. Enron email dataset. Available at: `https://www.cs.cmu.edu/~./enron/`, 2011.

[19] Ashley Davis. Implementing a flowchart with svg and angularjs. `http://goo.gl/REGJwH`.

[20] Paul Dietz and Darren Leigh. Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 219–226, New York, NY, USA, 2001. ACM.

[21] DocumentCloud. Backbone.js. `http://backbonejs.org/`.

[22] Yrjö Engeström. Interactive expertise: Studies in distributed working intelligence. *Department of Education, University of Helsinki*, 83, 1992.

[23] Piero Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, September 1999.

[24] Ben Fry and Casey Reas. Processing. http://processing.org/. Accessed: 22 Dec 2011.

[25] Ben Fry, Casey Reas, and John Resig. Processing.js. Available at: `http://processingjs.org/`.

[26] George W Furnas. *Generalized fisheye views*, volume 17. ACM, 1986.

[27] J. J. Garrett. Ajax: A New Approach to Web Applications, 2005. Adaptive Path LLC.

[28] Globalytica. Globalytica Software Tools: Te@mACH. `http://www.globalytica.com/thinksuite-html/`. Accessed: 2014-09-03.

[29] Google. Angularjs. `http://angularjs.org/`.

[30] Google. Npapi deprecation: developer guide. Available at: `https://www.chromium.org/developers/npapi-deprecation`, Retrieved: August 2015.

[31] J Richard Hackman. *Collaborative intelligence: Using teams to solve hard problems*. Berrett-Koehler Publishers, 2011.

[32] Avi Halachmi. Bug 888304 - content touch-events on Firefox-desktop should be disabled until we can support them properly (accessed 29 aug 2015). `https://bugzilla.mozilla.org/show_bug.cgi?id=888304`, 2015.

[33] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 115–118, New York, NY, USA, 2005. ACM.

[34] Richards Heuer. Private Correspondence, 2010.

[35] Richards J. Heuer Jr. *The Psychology of Intelligence Analysis*. Center for the Study of Intelligence, Central Intelligence Agency, Washington, DC, USA, 1999.

[36] Richards J. Heuer Jr. and Randolph.H. Pherson. *Structured Analytic Techniques for Intelligence Analysis*. CQ Press, 2010.

[37] MongoDB Inc. Mongodb. `https://www.mongodb.org/`, 2015.

[38] Alfred Inselberg. Multidimensional detective. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 100–107. IEEE, 1997.

[39] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proceedings of the 1st conference on Visualization '90*, VIS '90, pages 361–378, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[40] Joyent. Node.js. `http://nodejs.org/`.

[41] Daniel Kahneman. *Thinking, Fast and Slow*. Doubleday Canada, Mississauga, ON, 2011.

[42] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. Tuio - a protocol for table based tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)*, Vannes, France, 2005.

[43] KineticJS. Kineticjs - enterprise class web graphics. Available at: `http://www.kineticjs.com/`.

[44] Oleg V. Komogortsev, Dan E. Tamir, Carl J. Mueller, Jose Camou, and Corey Holland. Ema: Automated eye-movement-driven approach for identification of usability issues. In Aaron Marcus, editor, *Design, User Experience, and Usability. Theory, Methods, Tools and Practice - First International Conference, DUXU 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011, Proceedings, Part II*, volume 6770 of *Lecture Notes in Computer Science*, pages 459–468. Springer, 2011.

[45] Zum Kuckuck. Telekom realitme information graphics. http://projects.zumkuckuck.com/realtime/.

[46] Uwe Laufs, Christopher Ruff, and Jan Zibuschka. Mt4j - a cross-platform multi-touch development framework. *CoRR*, abs/1012.0467, 2010.

[47] Jing Li, Jean-Bernard Martens, and Jarke J. van Wijk. Judging correlation from scatterplots and parallel coordinate plots. *Information Visualization*, 9(1):13–30, March 2010.

[48] Zimu Liu, Yuan Feng, and Baochun Li. When multi-touch meets streaming. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia*, MUM '11, pages 23–32, New York, NY, USA, 2011. ACM.

[49] Lori Lorigo, Maya Haridasan, Hrönn Brynjarsdóttir, Ling Xia, Thorsten Joachims, Geri Gay, Laura Granka, Fabio Pellacini, and Bing Pan. Eye tracking and online search: Lessons learned and challenges ahead. *J. Am. Soc. Inf. Sci. Technol.*, 59:1041–1052, May 2008.

[50] N. Marquardt, R. Diaz-Marino, S. Boring, and S. Greenberg. The Proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies. In *ACM Symposium on User Interface Software and Technology - UIST'2011*, page 11 pages, Santa Barbara, CA, October 16-18 2011. ACM Press.

[51] Meteorpedia. Ddp clients. Available at: `http://meteorpedia.com/read/DDP_Clients`, Retrieved: August 2015.

[52] Miran Mirza, Jeff Wilson, and Robert Biddle. Collaborative annotations for large touchscreen web applications (in submission). In *3rd Workshop on Programming for Mobile & Touch*. ACM, 2015.

[53] Patrick Mueller. Web inspector remote. Available at: `http://people.apache.org/~pmuellr/weinre/docs/latest/`.

[54] Christian Müller-Tomfelde. *Tabletops - Horizontal Interactive Displays*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[55] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[56] Jakob Nielsen and Robert L. Mack, editors. *Usability Inspection Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[57] Nokia. Qt - cross-platform application and ui framework. `http://qt.nokia.com/`.

[58] NUITeq. Snowflake suite. `http://www.nuiteq.com/multitouchsoftware.php`.

[59] Palo Alto Research Center. Analysis of competing hypotheses software version ach2.0.5, 2010. `http://www2.parc.com/istl/projects/ach/ach.html`.

[60] David Pinelle, Carl Gutwin, and Saul Greenberg. Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Trans. Comput.-Hum. Interact.*, 10(4):281–311, December 2003.

[61] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International Conference on Intelligence Analysis*, volume 5, pages 2–4, 2005.

[62] C. Reas and B. Fry. *Getting Started with Processing*. Make Series. O'Reilly Media, 2010.

[63] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design: Beyond Human - Computer Interaction*. Wiley Publishing, 3rd edition, 2011.

[64] Alex Russell. Comet: Low latency data for the browser, 2006. http://alex.dojotoolkit.org/?p=545.

[65] Daniel M Russell, Mark J Stefik, Peter Pirolli, and Stuart K Card. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 269–276. ACM, 1993.

[66] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society.

[67] Herbert A Simon and William G Chase. Skill in chess: Experiments with chess-playing tasks and computer simulation of skilled performance throw light on some human perceptual and memory processes. *American scientist*, 61(4):394–403, 1973.

[68] Peter Simonyi, Jeff Wilson, Judith Brown, and Robert Biddle. Supporting "what-if" in touch-screen web applications (in submission). In *3rd Workshop on Programming for Mobile & Touch*. ACM, 2015.

[69] David Canfield Smith, Charles Irby, Ralph Kimball, and Eric Harslem. The star user interface: an overview. In *Proceedings of the June 7-10, 1982, national computer conference*, AFIPS '82, pages 515–528, New York, NY, USA, 1982. ACM.

[70] Diseree Sy. Adapting usability investigations for agile user-centered design. *Journal of Usability Studies*, 2(3):112–132, May 2007.

[71] Michiaki Tatsubori and Toyotaro Suzumura. Html templates that fly: a template engine approach to automated offloading from server to client. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 951–960, New York, NY, USA, 2009. ACM.

[72] Greg Trubulenz. Touchable. `https://github.com/dotmaster/Touchable-jQuery-Plugin`.

[73] Christophe Viau. Scripting inkscape with d3.js. Available at: `http://christopheviau.com/d3_tutorial/d3_inkscape/`.

[74] Ulrich von Zadow. libavg. Available at: `http://https://www.libavg.de/site/`.

[75] Ulrich von Zadow, Florian Daiber, Johannes Schöning, and Antonio Krüger. Globaldata: multi-user interaction with geographic information systems on interactive surfaces. In *ACM International Conference on Interactive tabletops and surfaces*, ITS '10, pages 318–318, New York, NY, 2010. ACM.

[76] Iwan Vosloo and Derrick G. Kourie. Server-centric web frameworks: An overview. *ACM Comput. Surv.*, 40(2):4:1–4:33, May 2008.

[77] Colin Ware. *Information Visualization, Third Edition: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2012.

[78] C. Wharton, J. Bradford, R. Jeffries, and M. Franzke. Applying cognitive walkthroughs to more complex user interfaces: Experiences, issues, and recommendations. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 1992.

[79] Laurie Williams, Robert R Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE software*, (4):19–25, 2000.

[80] Jeff Wilson. Tuiohost. `https://github.com/acc3ptC/tuiohost`, 2012.

[81] Jeff Wilson. Tuio express. `https://github.com/rjpw/tuio-express`, 2015.

[82] Jeff Wilson, Judith M. Brown, and Robert Biddle. Ach walkthrough: A distributed multi-device tool for collaborative security analysis. In *Proceedings of the 2014 ACM Workshop on Security Information Workers*, SIW '14, pages 9–16, New York, NY, USA, 2014. ACM.

[83] Jeff Wilson, Judith M. Brown, and Robert Biddle. Interactive parallel coordinates for collaborative intelligence analysis. *School of Computer Science Technical Report TR-14-05, Carleton Uni.*, 2014.

[84] Jeffrey Wong and Jason Hong. Marmite: end-user programming for the web. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1541–1546, New York, NY, USA, 2006. ACM.

Appendices

Appendix A

Illustrative Blog Posts

The following posts are available at my research project blog (rjpw.github.io), established for sharing lessons learned and design concepts. They are included here to illustrate some of the architectural transitions required to support proposed more advanced features of ACH-W (V2) and other potential sensemaking tools.

RESTful API Prototyping

RESTful API Prototyping

02 Feb 2015

Problem

When building web application prototypes there is value in rapid redesign on both clients and servers. Can we make it easy for client programmers to tinker with server prototypes?

Context

While in many situations it is possible to create a mock server API for clients to consume, in some cases there is value in more realistic data sources. Examples include real-time collaboration and Internet-of-Things (IoT) scenarios.

The situation can arise in small agile teams whose focus is primarily on web client applications, but who depend on evolving or establishing server-based resources and APIs. This may be early in proof-of-concept phases, after paper prototyping and perhaps in place of Wizard-of-Oz testing, and hopefully long before a production implementation.

Another context may simply involve the rapid provision of a more realistic *sandbox* API server that can stand in for or provide controlled access to expensive or sensitive resources.

Solution

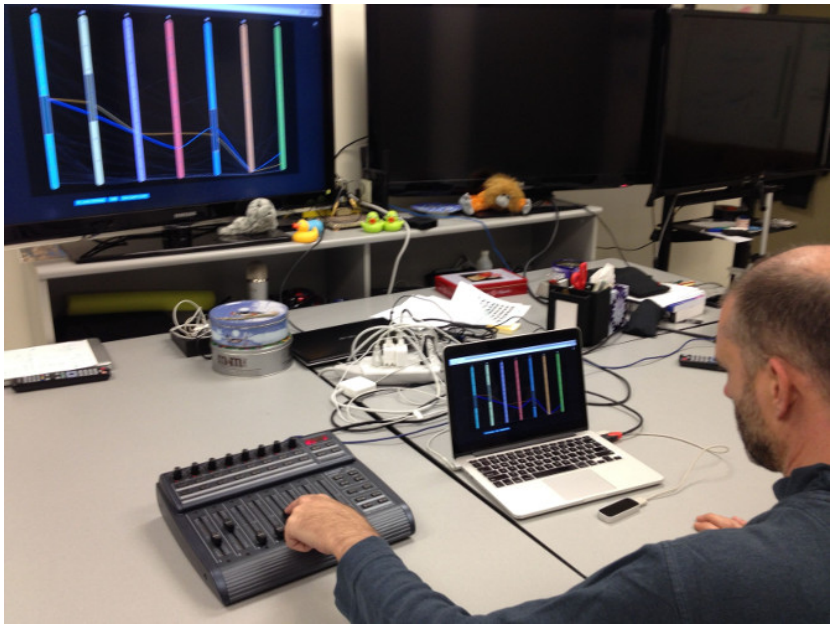
One solution that is experiencing a re-emergence of popularity and interest driven by the IoT community is *data flow programming* as exemplified by Node-RED, a visual editor from IBM Emerging Technologies. Node-RED and similar projects (see Resources at the end of this article) offers a number of advantages in API design over *mock*-based strategies or rapidly coding an API prototype from scratch, not the least of which is an interactive visual formalism of programmatic data flows.

Forces

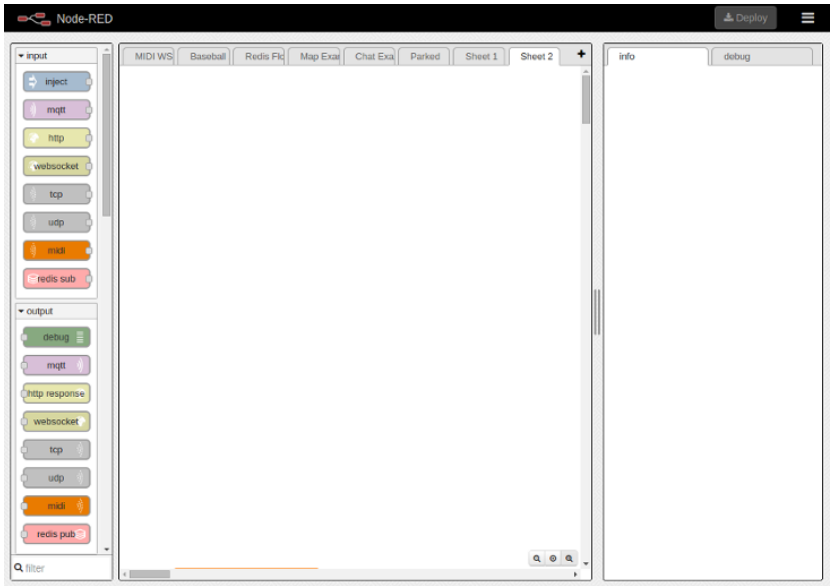
- Co-evolving server and client architecture or API
- Need for user validation of designs that goes beyond Wizard-of-Oz approaches
- Value in a self-describing architecture with a visual representation
- Need for rapid and moderately high-fidelity emulation of an actual resource

Example (Part I) - RESTful API for MIDI Controller

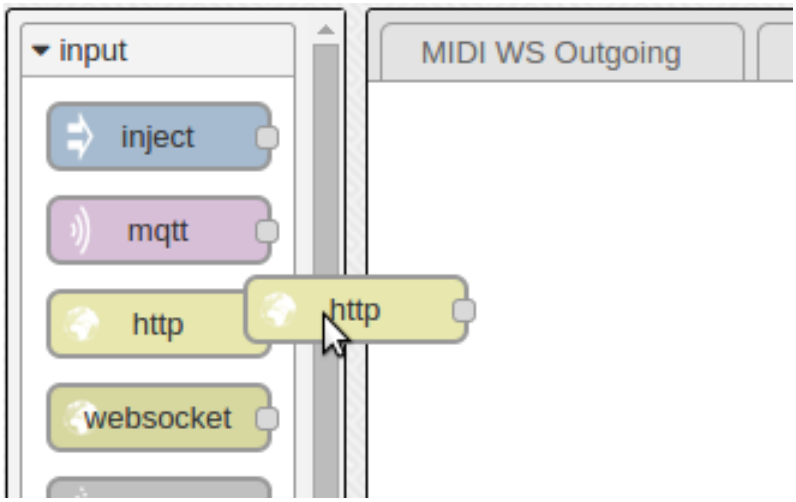
For our upcoming example, we will design a RESTful API for a motorized Behringer USB/MIDI Controller. Note that REST stands for Representational State Transfer, a design strategy that employs the HTTP protocol for server APIs based on resource names and a small set of simple operations, e.g. GET, PUT, POST, DELETE. We'll use the GET command to allow local and remote users to read the control positions, and we will use the PUT command to let local users update the MIDI values (remote users attempting to PUT will see "Permission Denied"). Ultimately we will use our API to control a parallel coordinates visualization.



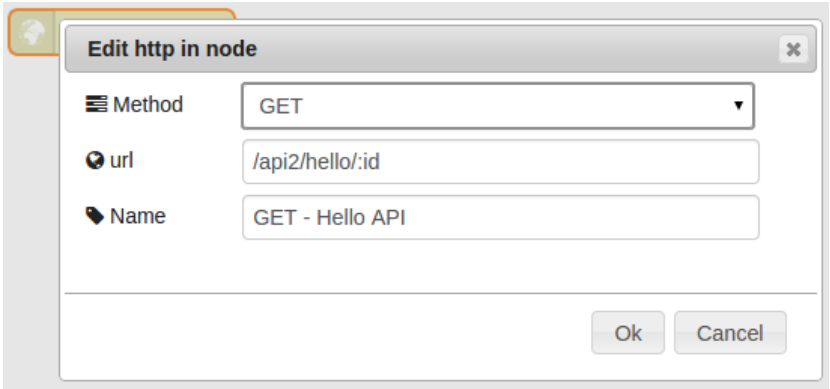
In my next post I will get into teaching Node-RED how to offer MIDI signals, but for the moment I will briefly explain how Node-RED works in general. The first time you open the flow editor you will see a blank workspace similar to the image below. In simple terms you write programs by dragging nodes onto the workspace, configuring them as necessary, connecting them together, and then hitting “Deploy”.



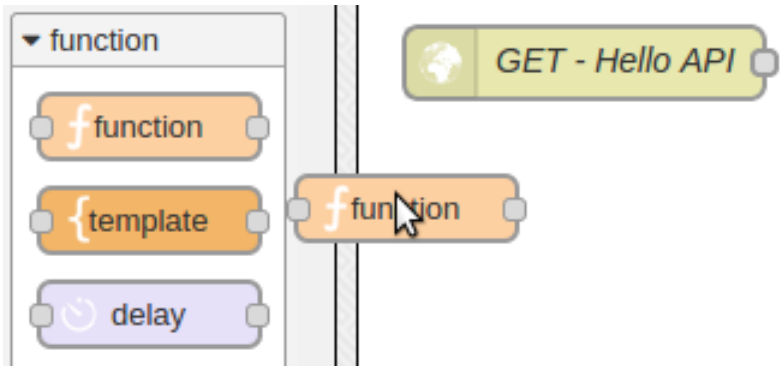
We start by dragging an HTTP input node onto the canvas.



Then we double-click on it to configure it. I defined the URL with a placeholder for “id”, and used the namespace “api2” to distinguish this walkthrough from an existing API on my system. Clearly you can name it whatever suits your needs.



Next we drag a function onto the canvas.

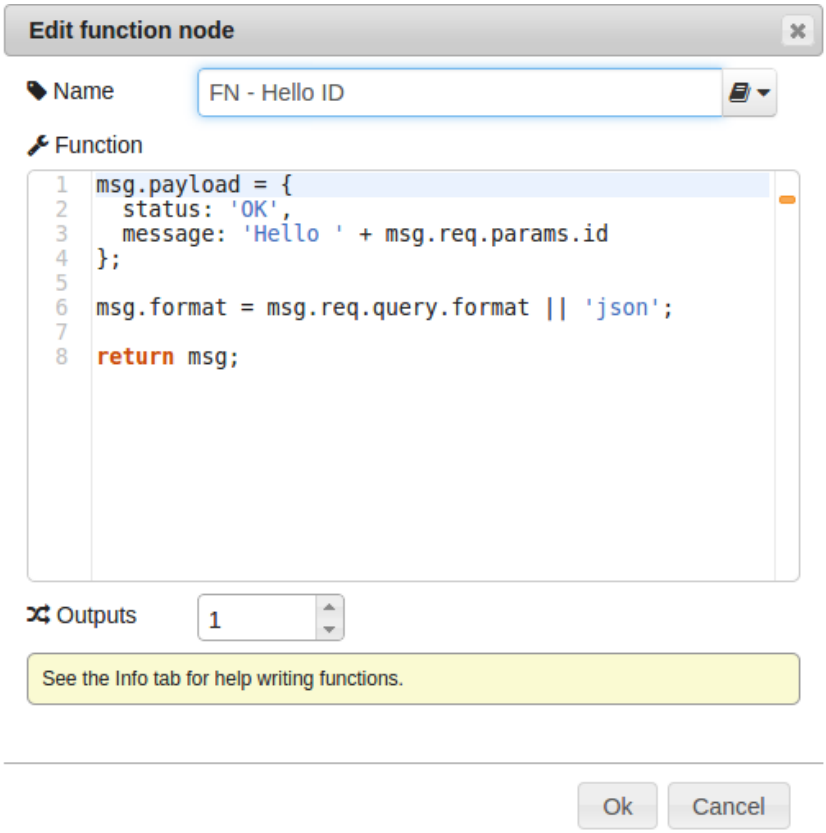


. . . and then we add a little JavaScript code. There are a few things to notice here. First, you might notice we can use the "msg" object immediately. This object is what is passed around your workflow, and you produce changes in your system by modifying its properties for downstream nodes. Most of the time you will be modifying msg.payload, as we do in this example, but notice you also have the ability to define new properties (msg.format in our example) that will be retained throughout the flow.

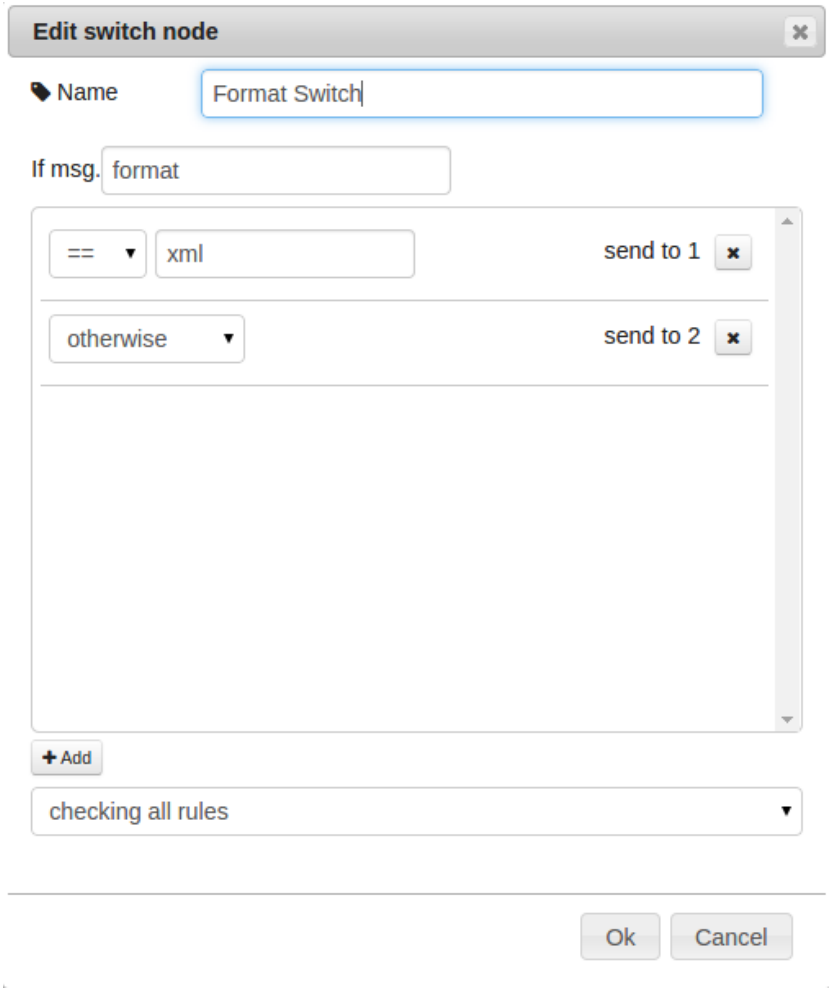
At this point in the workflow we are receiving input from an HTTP node, and so we can expect "msg" to contain a "req" object from Node-RED's embedded Express engine (you can get more details by clicking on the HTTP node and looking at the Info tab in your flow editor). We are going to be using req.params and req.query to express our API variables. Our *id* variable will come from req.params.id, and our *format* variable will come from req.query.format. Note that this is merely illustrative, and not necessarily a best practice.

We want to use the variable "msg.format" for a downstream Switch node, so we define it as a new property. We make sure to "return msg" so that our changes

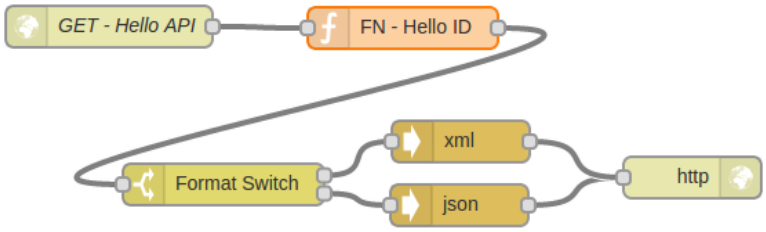
are passed along to the node's output.



We have decided to support both XML and JSON in our API. We could have used our input function to define two outputs, but there's an easier way. Node-RED comes with nodes that simply apply transformations to what they find in *msg.payload*. To create two streams, we will branch with a Switch node by dragging one onto the canvas and configuring it as shown. You can define as many outputs as you want. In our case we need two: one for XML and the other (the default case) for JSON.



Finally we connect the outputs of the switch to the inputs of the XML and JSON output nodes (no configuration required), and then their output is in turn fed into an HTTP output node. The final workflow looks like this.



We can now query our API from the console. First I will specify id = "readers", and format = "xml":

```
curl http://localhost:1880/api2/hello/readers?format=xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
  <status>OK</status>
  <message>Hello readers</message>
</root>
```

Recall that we had defined a JavaScript object in *msg.payload*, with properties *status* and *message*, and now these are serialized as an XML string.

To test the other format, this time I will specify format = "json" at the command line:

```
curl http://localhost:1880/api2/hello/readers?format=json
```

```
{
  "status": "OK",
  "message": "Hello readers"
}
```

The output is a JSON serialization, as requested.

Node-RED has already saved this into your personal workspace, and it will be there the next time you start the server. If you want to share this workflow with others you can easily serialize it by using their export function. This is what that looks like, a little "prettified" for this blog post. Try copying and pasting it using the "Import from ... Clipboard" feature of Node-RED (found under the "double hamburger" icon, upper-right), but please note that you will need to make it a little less pretty by removing the carriage return in the function definition in the second node. Apparently we paid a price for prettification.

```
[{
  "id":"d644025b.29bc", "type":"http in", "name":"GET - Hello API",
  "url":"/api2/hello/:id",
  "method":"get", "x":116, "y":85,
  "z":"1cb89973.e34767", "wires":[["b11e7de3.4ee18"]]
},
{
  "id":"b11e7de3.4ee18","type":"function","name":"FN - Hello ID",
  "func": "msg.payload = {\n  status: 'OK',\n  message: 'Hello ' + msg.req.par
  \n\nmsg.format = msg.req.query.format || 'json';\n\nreturn msg;",
  "outputs":1, "x":329, "y":86,
  "z":"1cb89973.e34767","wires":[["eeaeef8a.11511"]]
},
{
  "id":"89d6536f.7629b", "type":"http response", "name":"",
  "x":539,"y":194,"z":"1cb89973.e34767","wires":[]
},
{
  "id":"eeaeef8a.11511", "type":"switch", "name":"Format Switch",
  "property":"format",
  "rules":[{"t":"eq","v":"xml"},{"t":"else"}],
  "checkall":"true", "outputs":2,"x":199,"y":199,
  "z":"1cb89973.e34767",
  "wires":[["56dc2ada.a923d4"],["48af0eaa.b750f"]]
},
{
  "id":"48af0eaa.b750f", "type":"json", "name":"",
  "x":370,"y":214,"z":"1cb89973.e34767",
  "wires":[["89d6536f.7629b"]]
},
{
  "id":"56dc2ada.a923d4","type":"xml","name":"",
  "x":371,"y":168,"z":"1cb89973.e34767",
  "wires":[["89d6536f.7629b"]]
}]
```


If you have gone ahead and downloaded Node-RED for yourself you'll notice that my blank workspace has some input nodes that don't ship with the default installation (all the ones so far definitely are there). If you were to copy and paste a workflow containing undefined node types, Node-RED would let you save and deploy your workspace but then it would produce console errors indicating which node types were missing.

New node definitions can be added by following the instructions here. We will eventually be doing that after we write our first prototype MIDI API. For a quick preview of interacting with MIDI in NodeJS, we will try to get our MIDI signals

via the NodeJS RtMIDI wrapper project node-midi. The following NodeJS console script gives us a good starting point:

```
var midi = require('midi'), input = new midi.input();
input.openPort(1); // ID determined from testing

input.on('message', function(deltaTime, message) {
  console.log('message : ' + message);
});
```

When moving a slider on the panel, we see a sequence of messages like this:

```
message: 176,81,3
message: 176,81,2
message: 176,81,1
message: 176,81,0
```

In the next post we'll see how to incorporate our script into a RESTful API.

Resources and Links

- NodeJS
- Node-RED
- Behringer BCF2000
- node-midi
- Wizard-of-Oz
- NoFlo
- Video: Flow-Based Programming Using JavaScript

Managing Visualization State

Managing Visualization State

02 Mar 2015

Problem

How best to manage the state of interactive browser-based visualizations.

Context

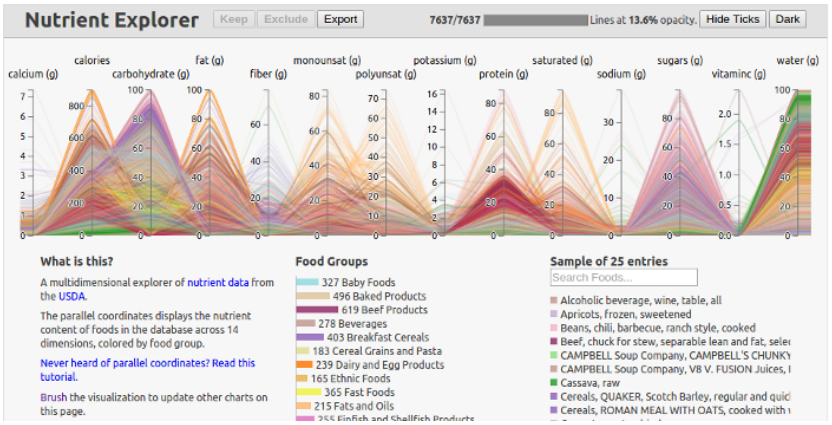
Interactive web applications composed of data-driven visualizations typically take advantage of open source JavaScript libraries. Choosing the right visualization library is one of the developer's first decisions, and an important but sometimes hidden element of this decision is how the library maintains interaction state. In the early days of the web it might be said that the DOM *was* the state, and many excellent visualization libraries start with this assumption. But with the rich forms of interaction and collaboration afforded by modern browsers, this assumption can lead to great complexity.

Often an application's features call for taking state away from the DOM and managing it entirely with JavaScript. Many libraries already contain their own abstractions for retaining user interaction, but their perfectly reasonable assumptions about event sources such as the mouse or touch-screen may be at odds with some requirements. Examples might include supporting undo/redo,

replaying interaction history, or synchronizing the interaction state of multiple visualizations.

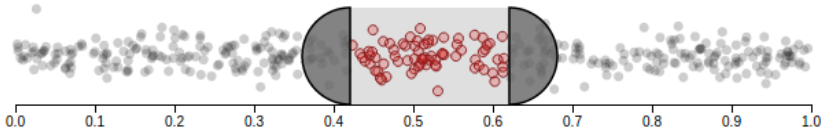
As of this writing, the most popular and powerful JavaScript visualization library is arguably D3, and there is a veritable encyclopedia of examples available on D3 author Mike Bostock's companion site bl.ocks.org.

To illustrate the notion of interaction state we borrow an example from the site that demonstrates a technique called brushing. Brushed visualizations can be used as a powerful visual query tool, and make it possible to achieve direct manipulation of query filters on related data. For an inspirational example see Kai Chang's Nutrient Explorer.



In a simpler example below from Mike Bostock on bl.ocks.org, brushing is used to define a subset of one dimension of the data. The selected values are highlighted by, in this case, applying or removing a CSS class that paints a dot red.

JavaScript-enabled users get an interactive version of this static image.



We can think of the data represented by the dots as the *domain model*, and the assignment of a CSS class based on brush position as retaining *interaction*

state. The state of the brushed region can be changed using mouse or touch (users of JavaScript-enabled browsers can interact with the example above), and when this happens the CSS class of each data point is re-evaluated. All other dimensions of this visualization are fixed at the point the page is refreshed.

But now we go beyond the intentionally simplified rendering in Bostock's example and ask what other potential degrees of freedom are possible in this one simple image. It is easy to imagine the underlying domain values changing, and when these values are returned we may find they exceed the current range, perhaps calling for the presentation of a scrollbar. A pinch gesture could perhaps expand or contract the range, or maybe we use a sliding window approach and pan horizontally using a swipe gesture. The long press of a mouse button could call up a magnifying glass, or the user could resize their browser window and the new ratio of width to height might trigger the presentation of a vertical axis and new vertical brush handles.

Each of these potential features would require associated state, the maintenance of which could be made local to the feature (as is the case above) or more usefully it could be held in a more central data structure to make it possible to inform other aspects of the page like the lists of food groups or specific dishes we see in Chang's Nutrition Explorer.

But these examples still only cover events local to the visualization. Imagine a system design where one user's brush arrangement represents just one submission in a larger collaborative real-time system, and before updating the actual display the user's individual input is averaged with those of other users connected to a server. Maybe our collaborative visualization is connected to a mix of virtual and physical devices (see earlier posts on RESTful API Prototyping and MIDI API), raising again the question of which collaborative data repositories or devices are considered sources of ultimate truth. In such cases the need for thoughtfully managing state becomes paramount.

Solution

When considering applications in these wider usage contexts it becomes useful to add an abstraction layer to hold onto state, and when we reach that point it's almost certain that our application could benefit from a client-side application framework. If understanding the code behind custom visualizations is already a challenge, blending it with frameworks like Backbone, Angular, or React can be all the more daunting. Here are a few key points to consider:

- Store interaction state centrally, but replicate only by necessity
- Maintain a strict separation between state and domain objects
- Strive for code that statelessly renders your views and visualizations
- Be selective in what gets rendered after a change to model or state

Forces

- Need for modes of interaction beyond simple clicks and drags
- Desire to leverage existing application and graphics libraries
- Potential need for multi-device collaborative input
- Bewildering abundance of choice in available development tools

Example (to come) - D3 meets React and Angular

In an example to come soon, we will combine D3 for visualization with React for stateless rendering and Angular for application coordination. For a preview of the concepts please follow this link to a video of a whirlwind coding session from Joe Maddalone of Egghead.io where he deftly pulls together exactly these technologies.

One might ask why use these three libraries. For several years now Google's AngularJS project has been helping developers build structured client-side applications with an MVC architecture. With so many live systems relying on AngularJS it seemed a helpful base to build on. As for Facebook's React library, their technique of rapidly rendering the "V" in MVC based on a virtual DOM and changes to state was an excellent way to meet our goal of stateless rendering. And of course D3 is the most widely used JavaScript library for authoring bespoke browser-based visualizations.

Even though each of these libraries is excellent for its own reasons, I must include a caveat that this particular combination of libraries/frameworks does not, on its own, guarantee satisfactory results. We might agree that pizza and gelato make a pleasant meal, but they probably don't make a pleasing mouthful. We'll need to ensure each of our libraries serves its own distinct role in the meal of MVC. We'll let AngularJS handle the model and controller in MVC, and we'll let React collaborate with D3 to produce the view. As for application state, we'll start by handling this with React for now, but we may end up writing a custom module later.

For a little more detail on the issues, we want to avoid letting both AngularJS and React manipulate the DOM because they take such a different conceptual approaches to rendering. AngularJS seeks to augment the standard *retained* mode view, whereas React seeks to ensure consistency and performance by employing an *immediate* mode approach that pre-renders and then adds or updates DOM elements (for a quick definition of retained and immediate modes see this MSDN page on graphics APIs, and for a conceptual rationale of the value of immediate mode in React see this video with Facebook's Pete Hunt).

But web browsers do render *documents* after all, not *applications*, and so ultimately both approaches are stuck with a stateful DOM. The React library lets us write our code *as if* we are re-rendering the DOM on every state change, but it does have code to deal with the state of certain DOM elements as we would

expect (e.g. it doesn't lose track of cursor position while updating textboxes, or scroll position while adding to lists). But in general it does what we were after by putting our own application state firmly under our control. In my experience there is also a performance benefit over AngularJS directives, and conceptually React components are much easier to understand.

For a little more background on our example, roughly three years ago the author of D3 wrote a blog post calling for new conventions for coding reusable charts. This eventually inspired another developer to suggest using React components to answer Bostock's request, asking in turn: D3 and React - the future of charting components? And in terms of providing an MVC architecture to consume external resources, we've already introduced Joe Maddalone's integration video at the top of this example where he shows how to get an AngularJS directive to delegate rendering to a ReactJS component, which in turn uses D3 to make a chart. In the next post we will put all these ideas together and then discuss where we've ended up.

Resources and Links

- Brushing and Linking
- Nutrient Explorer
- Integrating Components with D3 and AngularJS
- D3 and React - the future of charting components?
- Introducing D3 - Book Chapter
- Retained Mode versus Immediate Mode
- D3.js