Submitted to ECSCW'97—For comment only.

A Tale of Two Toolkits: Exploring the Relationship Between Infrastructure and Use

Paul Dourish
Apple Research Labs, Cupertino, USA
jpd@research.apple.com

W. Keith Edwards
Xerox Palo Alto Research Center, Palo Alto, USA
kedwards@parc.xerox.com

Abstract. Software toolkits for collaborative applications aim to provide generic, reusable components, applicable in a wide range of circumstances, that software developers can assemble to produce new applications. However, the nature of CSCW applications and the dynamics of group interaction present a problem for traditional approaches. Interaction in group settings is significantly influenced by the structure of the underlying infrastructure, in ways which toolkits typically do not support, and often exacerbate. This paper discusses these problems in the context of two specific toolkits, Intermezzo and Prospero. These systems were developed independently, but share a common design focus on the problems of flexibility. We outline some of the issues surrounding flexibility in toolkit design, illustrate potential solutions implemented in each of the systems, and comparatively discuss the approaches used.

Introduction

In the early days of collaborative system design, the software components for multi-user interaction and collaboration were crafted by hand for each application. Object replication and sharing infrastructures, access control facilities, consistency management mechanisms and multi-user interface widgets were written from scratch to

produce experimental systems for investigating the use of computer tools supporting collaborative work.

In this regard, CSCW development is no different from any other area of experimental system design. The consequence of being a pioneer is that, since no-one else has been there before, there is nothing left over from previous trips to be reused. Nor is CSCW different from any other area in the path of subsequent development when the needs of a novel application area are better developed. Common patterns of system design emerge, and developers begin to identify the core system elements and requirements which the application domain defines. As these become better understood, toolkits of reusable, widely applicable components emerge.

A cursory look at the historical development of CSCW applications and systems shows this trend. Most systems described at the first CSCW conference in 1986 (such as those described by Foster and Stefik (1986) or Grief and Sarin (1986)), or in the earlier workshop in 1984, were designed from scratch; however, by the third ACM conference in 1990, toolkits such as MMConf (Crowley et al., 1990) and Rendezvous (Hill et al., 1994) had emerged which offered programmers components which had been found to be generally applicable in a range of situations. Since then, new toolkits exploring new issues and techniques for application development have continued to appear (e.g. Trevor et al., 1995; Roseman and Greenberg,1996). Overall, the goal of toolkit development is to provide a set of facilities which are generic enough to be applied in a wide range of situations and yet specific enough to support the needs of different particular application situations.

In this paper, our concern will be with this balance between *genericity* and *specificity*. In particular, we will argue that the concerns of flexibility in cooperative work place particularly rigorous demands upon the designers of CSCW toolkits, and require novel techniques in the creation of toolkits.

Our argument sets up connections between a set of issues that are generally left unrelated. Discussions of issues in the design of CSCW toolkits or collaborative infrastructures are typically concerned with the generality of techniques for a range of application needs. Discussions of the flexibility of patterns of collaborative work are typically focussed on the features of particular applications.

By contrast, we are concerned here with a three-way relationship, from the range of behaviours in which collaborating users engage, through the facilities and mechanisms presented to them in applications, to the nature of the toolkit designs out of which these applications are constructed. The flexibility of the toolkit facilities that are presented by toolkit designers to application developers constrain the range of application features that can be designed, and these in their turn affect the ways in which collaborative work can proceed. All three are parts of the same equation.

We will illustrate these issues and potential solutions with reference to two particular toolkits that we have designed and implemented. Intermezzo (Edwards, 1996a) is a toolkit designed to support the *coordination* aspects of collaboration: the tasks associated with rendezvous of participants, awareness, and policy. A principal

element in the toolkit's design is that it gathers and exposes information about user activity to applications. Further, applications can modify the behavior of the toolkit based on this activity information. Prospero (Dourish, 1996a) deals largely with the areas of distributed data management and consistency control. It exploits an architectural approach called "Open Implementation" (Kiczales, 1992; 1996) in which the abstractions and mechanisms offered by a system (such as a toolkit) can not only be used by its clients (applications), but can also be examined and manipulated, and hence adapted to the needs of particular situations. Intermezzo and Prospero were developed independently, and take different approaches to the problem of toolkit flexibility, but they have a number of interesting features in common. First, they are motivated by many of the same concerns with the interaction of toolkit structure and application use; and second, they share a common technical concern with flexibility through specialisation rather than through genericity. Our goal here is to explore these concerns by comparing and contrasting the ways in which the two toolkits differently respond to a common set of design issues.

We will begin by discussing the problem of flexibility in CSCW, using a range of studies of cooperative group work to highlight the way in which collaborative work is a highly flexible and fluid activity. Our concern is not simply to point to the flexible accomplishment of group tasks, which we take to be, by now, a well-established observation. Rather, we want to illustrate that the fluidity of group work has significant consequences for the way in which collaborative systems are designed. Fluidity in action is not simply a matter of usage patterns, but rather is conditioned by (and must influence) the structure of the cooperative systems through which group work is conducted. We will go on to discuss flexibility in CSCW toolkits, and then to explore the interaction of use and infrastructure in more detail, drawing out implications for the design of CSCW toolkits. We will use examples from Intermezzo and Prospero to illustrate these ideas and show how these toolkits have taken different but complementary approaches to solving them. Next we step back to discuss more generally potential avenues of investigation for, first, the mutual implications of cooperative work and cooperative system design and, second, the design of flexible toolkits to support the implementation of cooperative systems.

Flexibility in CSCW

The need for flexibility is arguably greater in multi-user applications than it is in single-user ones. Multi-user applications bring with them all of the "traditional" single-user problems of adapting to user needs and accommodating individual differences—although of course, in the multi-user case, these problems are compounded by the fact that systems must be able to adapt to entire groups of users. But multi-user systems also must address new problems that arise from the fact that collaborative systems become essentially social venues, in which flexible management of issues such as privacy and awareness must be addressed. This section

discusses the need for flexibility in collaborative work, the types of flexibility that are required, and approaches to providing that flexibility in CSCW toolkits.

We will talk first about "use issues"—that is, studies of how collaborative activity proceeds and collaborative applications are used. We will then discuss "infrastructure issues"—that is, the range of ways in which toolkit facilities configure the computational substrate that defines and supports collaborative applications. Subsequently, we will explore how these two sets of concerns are inter-related.

Flexibility in Collaborative Application Use

Studies of cooperative working have repeatedly emphasised the fluidity and flexibility of coordination and activity.

Dourish and Bellotti (1992) report on a study of the use the ShrEdit shared text editor (McGuffin and Olson, 1992) in cooperative design tasks by small groups in an experimental setting. In comparison with a number of other cooperative writing tools of the same period, ShrEdit is highly unstructured. The group can share any number of textual documents, in which each author has an individual insertion point, and in which any number of authors can be working concurrently. ShrEdit employs an implicit locking mechanism, but locks at the level of characters, so that authors only come into conflict when two of them attempt to place their edit point at exactly the same character in the document. Changes made by any author are reflected almost immediately in the windows of the other authors (presuming that their view is scrolled to include the areas where the others are working), so that they have real-time access to each other's work.

Dourish and Bellotti's observations repeatedly emphasise the fluid and self-organising nature of the cooperative group activity engaged in by a number of sets of authors. Far from finding the lack of structured support for the cooperative writing process problematic, the authors would negotiate and manage the structure of their collaborative process in a natural and straightforward way. Groups varied in how they did this; some took separate responsibility for different parts of a single, long document, some worked largely in their own documents and then integrated their work towards the end of the experimental period, and others would essentially engage in free-for-all activity over the whole document. Perhaps more importantly, no group worked solely in any one way; rather, the "shared feedback" that the synchronous shared workspace provided to the group as a whole acted as a resource both for the individuals and for the groups as a whole to manage the group process on a moment-by-moment basis. This shared context allowed them to respond to the immediate circumstances of their work, build on what they could see each other doing, and negotiate, both explicitly and implicitly, the informal division of labour by which their activity was organised.

Beck and Bellotti (1993) report on other experiences of co-authoring, in quite different settings; the primary collaboration described was the naturally-occurring collaborative writing of an academic conference paper by two authors separated by

six thousand miles and eight time zones. (Elsewhere, Beck (1994) has reported on other studies of collaborative authoring that provide further supporting evidence for the observations made in this study.) A significant finding in this work was that, while the authors would establish a division of labour and a plan for the coordination of their joint work over the document, this pre-arranged plan would be regularly *and unproblematically* ignored in the actual performance of the work. That is, while the authors might divide up responsibility for two sections of document, they would each, in fact, make contributions to the other's section, whether to correct typos or to add substantive material relevant to their own sections, as relevant. In fact, Beck and Bellotti observe, the success of the collaborative process may in many cases depend on just this sort of "opportunistic" action that successfully advances the group's work while failing to uphold the separation of individual activities.

Flexibility in CSCW Infrastructure

Just as flexibility emerges in all aspects of the use of collaborative technologies, approaches to toolkit design have also been faced with a bewildering array of areas in which technical configuration must be flexible in order to support the variety of mechanisms which might be required in client applications. We briefly consider a number of these here.

Data distribution concerns the ways in which the computational representations of data available to participants (the text in a collaborative writing system, the records in a multi-user database or the marks on a shared whiteboard) will be located, found, accessed and moved in a collaborative system. For example, a single copy of any data item may be located on a central server, or multiple copies may be accessible at each node in the network. Data objects may be stored at fixed locations through the course of a session or collaboration, or may move around according to the pattern of access which emerges. Access to this distributed data must be regulated so as to maintain consistency in each user's view. Such *consistency control* may be achieved through regulating access to the shared data store, or by requiring clients to obtain locks on data items, or by outlawing sequences of action which might result in inconsistency. *Access* to the data store may be governed by permission settings, by roles, or by opportunity. *Sessions* may arise simply through the individual actions of collaborators, or may be set up explicitly, and might be managed on an invitation basis, or regulated by an individual.

Dourish (1995a) outlines three aspects of flexibility in the design of CSCW systems. *Static* flexibility refers to the extent to which the system can support a variety of collaborative needs (such as "synchronous" or "asynchronous" work[1]). *Dynamic* flexibility reflects a system's ability to respond to the changing circumstances of a collaborative session (such as changes in group membership). *Implementation* flex-

1. We do not believe these terms are unproblematic—hence the "scare quotes"—but use them to refer to a set of relatively well-understood application usage situations recognised in the literature.

ibility refers to the ability of a system to operate over a range of implementation substrates (such as different forms of network or network topology, or different approaches to data store management). To one extent or another, each of these is a concern for the development of any toolkit, just as a failure of any, in a toolkit or an application, can itself move from being an infrastructure problem to one of use. This issue, the interaction of use and infrastructure, is a critical motivation for our approaches.

The Interaction of Use and Infrastructure

Collaboration by definition involves the activities of multiple users, the presence of whom affects the nature of the systems we build. On the systems side, our applications and toolkits must be cognizant of multiple input streams, must address consistency control, and may potentially have to deal with problems of distribution.

The presence of multiple users also has implications for the social aspects of the systems we create. A multiuser application becomes a medium for interpersonal exchange, with all of the potential concerns (as well as benefits) which that entails (Bentley and Dourish, 1995). Examples of the more socially-oriented aspects of collaborative systems that must be addressed include privacy of participants, awareness, and support for the rapidly shifting roles of the participants in the collaboration.

However, these concerns—the "social" and the "technical"—are typically explored in isolation; not just in different papers, but in different rooms. Toolkit designers focus primarily on the range of applications they wish to support in motivating and evaluating the flexibility of their toolkits. Those engaging in user studies or studies of collaborative work settings typically do this with reference to the facilities provided by specific applications or the requirements for new ones. The focus of the toolkit designer suggests that applications can be thought of independently of the situations in which they will be used; while those studying collaborative activities suggest, inversely, that applications can be studied independently of the toolkits and infrastructure facilities which give rise to them.

Our concern, then, is not with the implications for applications of flexible patterns of group work, nor with the implications for application development of flexible infrastructure. Rather, we are primarily concerned with the direct interaction between use and infrastructure, and its consequences.

For example, the decision of a particular toolkit to provide locking, or optimistic serialization, or some other concurrency control mechanism affects the style of use of the applications built with that toolkit[2]. A toolkit that requires that objects be explicitly locked before they can be manipulated enforces a heavyweight style of interaction on its applications; this style of interaction may be manifest to the users

2. We will discuss approaches to consistency control in more detail later.

of the system as well, by requiring them to engage in interface activities which reflect this heavyweight coordination. Greenberg and Marwood (1994) have examined a range of such concurrency control techniques and their influence on interaction.

Consider as a second example a toolkit that uses the notion of roles to establish access control for data objects. In a shared text editor application, these roles might include editor, author, and commentator. Such roles codify a set of social practices that exist at a certain moment in time, but are not easily adaptable to new situations and cannot accommodate the moment-to-moment shifting interactions which characterize so much of interpersonal communication. Previous work by Dewan et al**.** (1994) and Neuwirth et al**.**(1990) has noted such problems.

Both of these examples illustrate how choices in infrastructure design can influence the user interface and multiuser "social interface" of our systems.

The point of these examples is not to say that certain styles of locking or access control are necessarily bad; rather, they show that any choice of implementation strategy can potentially influence application functionality and hence usage. Any toolkit designer endeavors to accommodate a wide range of applications. But designers of toolkits for collaborative systems must take special care they not only allow but *support* the construction of applications that are responsive to the fluid styles of interaction required by collaboration.

Approaches to CSCW Toolkit Flexibility

This section describes the approaches taken by our two toolkits—Intermezzo and Prospero—to provide flexibility to application writers. While these systems have a common goal of flexibility, they approach this goal from very different directions. Intermezzo uses an approach based on reifying the setting in which a collaboration occurs, and allowing applications to alter the behavior of the toolkit based on situational context. The Open Implementation approach used in Prospero exposes aspects of toolkit-internal constructs to application writers. These constructs can then be specialized and reused by the toolkit.

Example: Flexibility through Awareness in Intermezzo

Intermezzo addresses flexibility by providing an infrastructure by which some of the technical aspects of collaborative applications can be mediated by input about the social setting in which the collaboration occurs. For example, a collaborative writing application may need to accommodate different styles of session management or access control during the course of a collaboration based on the shifting goals and needs of the participants. To support this form of dynamic flexibility, Intermezzo brings information about the participants of the collaboration, their activities, and their environment into the realm of the toolkit. This information is

made available to applications (modulo privacy restrictions) in a machine-parsable format. Further, the toolkit itself uses this information about situational context to regulate its internal operations.

Intermezzo relies on information about the state of the world that is represented as a database of objects describing user activity. Representations of activity are hierarchical and allow application-specific "views" of the world state at any number of semantic levels. Activity information has a structured format and supports links between related objects. In the Intermezzo model, applications themselves are responsible for maintaining the global view of the world, and the toolkit provides support (much of it automatic) for publishing and updating this information. For example, when an Intermezzo-based collaborative writing tool is started, it will publish information describing itself (the tool), its users, and the documents being edited. The format of the published data allows it to be searched, updated, and viewed easily by other applications.

The principal way in which Intermezzo provides flexibility to applications is by allowing them to adapt not just their own behaviour, but also the behavior of the toolkit in reaction to the changing dynamics of the world in which they are run. Applications change toolkit behaviour in two ways. The first is by downloading code into the runtime system that runs in response to changes in world state. This downloaded code can be tied to any change in the environment, and can directly affect application or toolkit state.

Downloaded code, which is written in an extended version of the Python language (Van Rossum, 1995), can be directly executed by the runtime system at the time it is transmitted; can be set to "fire" when a particular change on a specific object occurs in the activity database; or can fire whenever the database state achieves a certain "pattern," as described via a pattern matching language. By associating portions of application code with objects "in the world," application behavior can be directed and influenced by changes in situational context.

The second way applications can change the behavior of the toolkit is through the use of situationally-based access control (Edwards, 1996b). Intermezzo uses strong, cryptographically-secure access control throughout. The objects to which access is controlled are not just the objects created by the application to maintain domain-specific application state, they also include objects used internally by the toolkit. By constraining access to toolkit functions and data, applications can effect global changes in the behaviour of the toolkit itself. For example, application code can provide situational control over inter-application behaviour such as session management through this technique (Edwards, 1994).

The actual access control settings of various objects—both in the application and in the toolkit—are under the control of a language-based policy system that uses information about the situational context as an input. Applications use the access control system by writing "policies"—which are essentially access control templates—and binding them to "roles." Unlike traditional roles, however, the

membership of an Intermezzo role is *described* by a predicate function, rather than *defined* by a membership list. This technique allows applications to be written in terms of general descriptions of user behavior and context, rather than more limiting specific definitions. For example, an application could respond to general description of "the set of people currently in my lab," rather than the more static and restrictive *a priori* enumeration of the names of specific people often found in my lab.

By mediating the systems-oriented aspects of access control, session management, and awareness with input about the context in which a collaboration is occurring, Intermezzo provides applications with *dynamic* flexibility—the ability to match and adapt to the changes in a group's interactions over time. Intermezzo enables applications to more easily support groups that move fluidly between various styles of work.

Example: Flexible Consistency Control in Prospero

Consistency control is the mechanism by which a collaborative system ensures that the potentially simultaneous actions of multiple users over a shared data space do not result in inconsistent views of the data space. For instance, consider a medical system supporting shared access to patient records. To ensure speedy response, the system might replicate the database, placing copies of the patient records at different points in the network. If two users were to simultaneously update records, and their changes were to be made to their own copies of the records before being sent across the network, then an inconsistency would have arisen, since their respective copies of "the same" record would display different information.

Since the shared spaces of collaborative systems can be modeled as databases, most CSCW systems have looked to the models provided in distributed database design for consistency mechanisms. The most common mechanisms in database design are forms of "locking", in which a "lock" for data to be modified is obtained before the changes can be made. The locking mechanism forces clients to declare their intent to update records before the updates are actually made, which provides an opportunity to avoid inconsistency by refusing to grant locks if the actions might conflict with others. If multiple clients simultaneously request "read locks" (locks obtained by clients which intend to read data), then they can all be granted, because there is no opportunity for inconsistency to arise. However, if one client has already been granted a "write lock" for a piece of data, then no other client will be granted a second "write lock" until the first client has finished, because two simultaneous writes might result in inconsistency—a write/write conflict. In many situations, an outstanding write lock might also prevent any further read locks being granted, because the reading client might receive data which is out of date.

The database model, with its read and write semantics, is very generic; it supports a wide range of potential collaborative activities. However, drawing on the interaction of use and infrastructure outlined above, we can identify a number of

problems. Locking out operations because of potential conflicts can interfere with the smooth progress of collaboration; and the read and write semantics are such that many activities look like conflicts even if, in fact, they will not lead to inconsistency. For example, consider two users working on a bibliographical database, which records citation details for publications. Adding two records at once will not lead to conflicts; either they're the same record (in which case, they match to the same new record to be added, which should be an acceptable operation), or they're different records (in which case, they should both be added). However, to the database substrate, these will look like two write operations, and so a conflict will be flagged. In other words, the configuration of the infrastructure, a database storage layer with a conflict avoidance mechanism based on read and write semantics, will interfere with the execution of collaborative work, forcing users to interleave their activities.

Prospero's approach is to allow consistency control to be specified in terms of the operation semantics of particular collaborative applications (Dourish, 1996b). The paradox, of course, is that consistency management is a toolkit concern, but the toolkit must be free of exactly the type of application features from which we want to construct this mechanism. The toolkit itself, then, does not talk in terms of application semantics (although it provides a reusable and extensible core set of potential properties applicable in some range of situations). Instead, it provides the framework within which they can be defined. Thus, the developer of the bibliographical database example would describe the semantic properties of application operations (such as the non-destructive writes implied by adding new entries to the database), and then the toolkit can manage consistency in these terms. The toolkit is specialised to the needs of the particular implementation.

Prospero uses a pairwise comparison of operation properties, much like the read/write comparison model, to decide when sets of operations might conflict. However, using the metalevel control (that is, the mechanism for programmers to modify or augment internal toolkit facilities) allows the programmer to gain control over this process and cast it in terms of the application specifics. Since the comparison is done in terms of operations which are meaningful at the application level, rather than simply those meaning at the infrastructure level, the toolkit can support a range of behaviours specific to each situation, which would be disallowed by a traditional approach implemented over a standard database model. In this way, a link is achieved between the configuration of the infrastructure and the particularities of each particular application.

The Design of Flexible CSCW Toolkits

Intermezzo and Prospero illustrate two different approaches to achieving flexibility in toolkit design. However, they share a common set of concerns, and in particular, a common reaction to the traditional mechanisms of toolkit flexibility.

The standard approach to flexibility in toolkit design (in any domain, not just CSCW) is through genericity. Genericity is the design of generic toolkit facilities, applicable to the widest range of applications. For instance, a standard database model, with access control and consistency management based on "read" and "write" access, is highly generic, and can therefore serve as an infrastructure for a very wide range of CSCW applications. Access to the shared data store can be implemented in terms of read and write operations, in whatever way the application might then use those reads and writes to support the specific application needs. As a generic mechanism, then, the traditional approach takes the read/write database model as a useful basis for the design of a toolkit which can be used to build CSCW applications of all sorts.

Intermezzo and Prospero take almost the *opposite* approach to the provision of flexibility in CSCW toolkits. Their primary concern is with the design of effective, functional CSCW applications, and so they are concerned with the highly specific behaviours observed in collaborative settings. As we have explored in this paper, these behaviours depend in crucial ways on the variety of infrastructural configurations within which cooperation takes place. The implication, then, is that applications require widely different infrastructure provisions, rather than a single highly generic mechanism onto which a variety of application requirements can somehow be "mapped". The question of flexibility in Intermezzo and Prospero, then, is how the toolkit can be specialised and adapted to the needs of collaborative settings, rather than how those settings can be described in terms of some universal base level.

Intermezzo primarily addresses *dynamic flexibility*: the ability of applications to track the changing circumstances in which they are run and adapt their behavior accordingly. The infrastructure mediates its own operation through application-supplied code that takes as input a repository of information about the application's context. Architecturally, Intermezzo takes a "programming language" approach, providing specialised languages in which the needs of specific applications can be described.

Prospero primarily addresses *implementation flexibility*: the ability of applications to "open up" the internal constructs of the toolkit and adapt it to their particular requirements. Applications can modify and specialize toolkit-internal constructs, which the toolkit will then use to provide application support. At the same time, it provides novel mechanisms such as the divergence/synchronisation approach to distributed data management (Dourish, 1995b) which generalise across the traditional boundaries of collaborative systems, as a means to address (by reformulating) issues of static flexibility. Architecturally, Prospero is based on the "Open Implementation" approach which makes aspects of the toolkit, traditionally only available for "use" by applications, amenable to examination, modification and control. (It is perhaps interesting that the Open Implementation approach is derived from work in the theory and design of programming languages. Perhaps this work

lends further support to the theory that Computer Science is a bell-shaped curve around programming language design and implementation.)

The different architectures, however, belie a shared underlying technical goal; to let application code "push semantics" into the toolkit to accomplish particular goals. This goal is motivated by both technical and social or observational needs. Technically, providing such a mechanism is a solution to the perennial problem of flexibility in toolkit design. Socially, semantic features (the "meaning" and behavioural consequences of toolkit configurations) are seen to be crucial in the emergence of forms of group behaviour, and so are ineliminably the concern of application developers and users, not of toolkit designers, who, by definition, are separated from the situations in which the toolkit mechanisms will be realised in particular applications and put to use in particular circumstances.

Conclusions

The days when CSCW applications were build by hand, from scratch, are long behind us. Not only is such an approach increasingly technologically impractical (as systems get larger and more complex, and as users demand more and more from the applications they use), but it is also inviable commercially. Like applications in any other domains, CSCW applications depend on toolkits to provide them with standard, reusable approaches to application design and mechanisms to be deployed. The critical concern for the designer of a CSCW toolkit is the range of behaviours (and hence of applications) which can be supported.

The design of toolkits, and the flexibility implied by the range of mechanisms they supply and the range of ways in which they can be combined, has typically been approached purely as a technical problem. However, the radical levels of flexibility implied by collaborative work, and observable in studies of groups working together, undermines this position. Observational studies have pointed to the range of ways in which collaborators organise (and reorganise) their work, and these studies illustrate how group behaviour is critically dependent not simply on the "high-level" facilities which collaborative applications offer, but also on the "low-level" issues of infrastructure configuration which lies beneath these applications.

These observations suggest that the problem of toolkit flexibility is not one of providing generic mechanisms, but rather is one of gaining control over the relationship between the application and the infrastructure. Semantic issues cannot be locked inside a toolkit, inaccessibly to the application designer and the user; rather, the semantics required of the toolkit come from applications and circumstances of use. In other words, the focus for design should be the three-way relationship amongst toolkits, applications and use, rather than the traditional pair of two-way relationships (toolkit/application and application/use).

We have outlined and motivated these problems by appeal to a number of studies, and shown how two toolkits, Intermezzo and Prospero, tackle these problems. Inter-

mezzo and Prospero were designed independently, employ different computational architectures and address different domains of concern to the designer of a CSCW toolkit. However, they share a concern with the relationship between application and infrastructure, and with making this accessible so that application developers can free themselves from the rigid models which are embodied (and often hidden) in CSCW toolkits. Intermezzo and Prospero both emphasise flexibility through application specialisation, rather than flexibility through abstraction and genericity.

The general concern which motivates these two systems, and the mechanisms which they embody, point the way towards a new model of toolkit design which is grounded not simply in the technical concerns of generic infrastructure mechanisms, but rather which is based in understandings of how collaboration works.

Acknowledgments

Prospero was designed and implemented while Dourish was employed at the Rank Xerox Research Centre (formerly EuroPARC) and studying at University College, London; Intermezzo was designed and implemented while Edwards was studying in the College of Computing at Georgia Institute of Technology. We would like to thank the voices that spoke to us and told us what to say. Some of these include Beth Mynatt, John Stasko, Dik Bentley, Jon Crowcroft, Beki Grinter, and Prasun Dewan for their comments and contributions.

References

Beck, E. and Bellotti, V. (1993): "Informed Opportunism as Strategy", in *Proc. Third European Conference on Computer-Supported Cooperative Work ECSCW'93* (Milano, Italy), Dordrecht: Kluwer.

Beck, E. (1994): "Practices of Collaboration in Writing and their Support." D.Phil. thesis, University of Sussex (Technical Report CSRP 340, ISSN 1350 3162).

Bentley, R. and Dourish, P. (1995): "Medium vs. Mechanism: Supporting collaboration through customisation", in *Proc. European Conference on Computer-Supported Cooperative Work ECSCW'95* (Stockholm, Sweden), Dordrecht: Kluwer.

Crowley, T. Milazzo, P., Baker, E., Forsdick. H. and Tomlinson, R. (1990): "MMConf: An Infrastructure for Building Shared Multimedia Applications", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'90* (Los Angeles, California), New York: ACM.

Dewan, P., Choudhary, R., and Shen, H. (1994): "An Editing-based Characterization of the Design Space of Collaboration Applications," *Journal of Organizational Computing*, 4(3), pp. 219-240.

Dourish, P. and Bellotti, V. (1992): "Awareness and Coordination in Shared Workspaces", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'92* (Toronto, Canada), New York: ACM.

Dourish, P. (1995a): "Developing a Reflective Model of Collaborative Systems," *ACM Transactions on Computer-Human Interaction*, 2(1), pp. 40–63.

Dourish, P. (1995b): "The Parting of the Ways: Divergence, Data Management and Collaborative Work", in *Proc. European Conference on Computer-Supported Cooperative Work ECSCW'95* (Stockholm, Sweden), Dordrecht: Kluwer.

Dourish, P. (1996a): *"Open Implementation and Flexibility in a CSCW Toolkit"*, PhD dissertation, Department of Computer Science, University College, London.

Dourish, P. (1996b): "Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'96* (Boston, Mass.), New York: ACM.

Edwards, K. (1994): "Session Management in Collaborative Applications," in *Proc. ACM Conference on Computer-Supported Cooperative Work* CSCW'94 (Chapel Hill, North Carolina), New York: ACM.

Edwards, K. (1996a): "Coordination Infrastructure in Collaborative Systems", PhD dissertation, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Edwards, K. (1996b): "Policy and Roles in Collaborative Applications", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'96* (Boston, Mass.), New York: ACM.

Foster, G. and Stefik, M. (1986): "Cognoter, Theory and Practice of a Collaborative Tool", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'86* (Austin, Texas), New York: ACM.

Greenberg, S. and Marwood, D. (1994): "Real-time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface", in *Proc. ACM Conference on Computer Supported Cooperative Work CSCW'94* (Chapel Hill, North Carolina), New York: ACM.

Grief, I. and Sarin, S. (1986): "Data Sharing in Group Work", in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'86* (Austin, Texas), New York: ACM.

Hill, R., Brinck, T., Rohall, S., Patterson, J., and Wilner, W. (1994): "The Rendezvous Architecture and Language for Multi-User Applications", *ACM Transactions on Computer-Human Interaction*, 1(2), pp. 81–125.

Kiczales, G. (1991): "Towards a New Model of Abstraction in the Engineering of Software", in *Proc. IMSA Workshop on Metalevel Architectures and Reflection*, Tokyo, Japan.

Kiczales, G. (1996): "Beyond the Black Box: Open Implementation", *IEEE Software*, pp. 6–11, January.

McGuffin, L. and Olson. G. (1992): *"ShrEdit: A Shared Electronic Workspace"*, CSMIL Technical Report, Cognitive Science and Machine Intelligence Laboratory, University of Michigan.

Neuwirth, C.M., Kaufer, D.S., Chandhok, R., and Morris, J. (1990): "Issues in the Design of Computer Support for Co-authoring and Commenting," in *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'90***,** pp. 183-195 (Los Angeles, California), New York: ACM.

Roseman, M. and Greenberg, S. (1996): "Building Real-Time Groupware with GroupKit, a Groupware Toolkit", *ACM Transactions on Computer-Human Interaction*, 3(1).

Trevor, J., Rodden, T. and Blair, G. (1995): "COLA: A Lightweight Platform for CSCW", *Computer Supported Cooperative Work*, 3, pp. 197–224.

Van Rossum, G. (1995): *Python Reference Manual Release 1.3***.** October 13 (available as http:// www.python.org/doc/ref/ref.html).