RC 21694 (97736) 03/20/2000
Computer Science / Mathematics 10 pages

IBM Research Report

A Shearing Layers Approach to Information Systems Development

Ian Simmonds
IBM T J Watson Research Center
PO Box 704
Yorktown Heights, NY 10598
simmonds@us.ibm.com

David Ing
IBM Advanced Business Institute
Route 9W, Palisades
NY 10964-8001, USA
daviding@ca.ibm.com

LIMITED DISTRIBUTION NOTICE
This report has been submitted for publication outside of IBM and will probably be copyrighted is accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Some reports are available at http://domino.watson.ibm.com/library/CyberDig.nsf/home. Copies may requested from IBM T.J. Watson Research Center, 16-220, P.O. Box 218, Yorktown Heights, NY 10598 or send email to reports@us.ibm.com.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

A Shearing Layers Approach to Information Systems Development

Ian Simmonds
IBM T J Watson Research Center
30 Saw Mill River Road, Hawthorne
NY 10532, USA
simmonds@us.ibm.com

David Ing
IBM Advanced Business Institute
Route 9W, Palisades
NY 10964-8001, USA
daviding@ca.ibm.com

ABSTRACT

In this paper we respond to the observation that systems are subjected to qualitatively different scales and rates of change, and should consequently be constructed to adapt in "shearing layers." This observation applies equally to social systems such as business (or other) enterprises, and to the software systems that they use.

Our response is multifaceted, concerned with three different objects of design: the design of two kinds of organization; the functional design of software applications to support those organizations; and the implementation of those applications as software. The two relevant kinds of organization are large information systems services organizations, and the organizations that use their services. When an organization is designed to respond to change it is called an adaptive enterprise. Software so designed is called adaptable software.

Among these three forms of design, this paper focuses on the middle one. We propose constructs called "agents" and "conversations" as means for reasoning about the functional design of adaptable software. The bridge to organization design comes from determining which people should engage in what conversations to achieve what forms of learning. The bridge to software implementation comes from determining how to deploy each conversation within available middleware and other technologies.

We do not wish to claim that shearing layers is a new phenomenon in information system development. Indeed, we present examples of practices consistent with this approach. Rather, we seek to name the phenomenon, present theory that embraces that phenomenon, and demonstrate how we are applying that theory within our ongoing work. As a result, this paper can be seen as a source not only of newly explicit requirements on software engineering, but also of some possible solutions.

1. BACKGROUND

> Blame for the systems development crisis has been laid at the feet of the creators of development methods, tool builders, analysts, designers, and implementors. But we suggest that the problem may, instead, lie in an incorrect goal set that we all have accepted from the outset that is the idea that systems should support organizational stability and structure, should be low maintenance, and should strive for high degrees of user acceptance.
>
> Truex, Baskerville and Klein, 1999 [40]

1.1 Adaptive Enterprises need Adaptable Software

In his book *Adaptive Enterprise* [20], Haeckel points to the effect on the strategy of a business enterprise of the increasingly frequent unpredictable, discontinuous changes that occur in the environments in which it operates. How many enterprises predicted the rise of the World Wide Web and e-commerce? How many *software* companies made this prediction?

The kinds of change observed by Haeckel and others are more than a temporary phenomenon. They are a consequence of the very moral foundations of commercial life. As Jane Jacobs has pointed out [25], the moral precepts that govern commercial life require that commercial organizations use initiative and enterprise in promoting comfort and convenience for their customers, and that in doing so they be thrifty, efficient and industrious. In other words, they are morally driven to actively seek to change (improve) what they do for their customers and how they go about doing it.

Above all else, frequent and disruptive technological, regulatory, political or competitive changes lead customers to reconsider just what they consider to be of value. As such they render traditional "predict and plan" approaches to business and information technology strategy not only inadequate, but downright dangerous to the long term health of any enterprise offering them products or services.

The only possible alternative, in Haeckel's opinion, is to focus strategic thinking on the design of an adaptive, learning organization. Haeckel suggests that organizational

learning and adaptivity are optimized when they are focused on two things [20]. Firstly, there should be constant inquiry into what customers consider to be of value. This requires that, at the very least, customer relationships be designed to ensure learning. Secondly, there should be constant inquiry into what organizational capabilities the enterprise must develop, informed by an understanding of what customers consider to be of value.

But while enterprises and their people may seek to become ever more adaptive, software is at best adaptable.[1] Adaptive enterprises need information system development processes that ensure that changes are made not only in a timely manner but are a creative response to changes both in organizational needs and technological possibilities. However these engineering approaches are still missing.

Note that adaptation is doubly interesting to an information systems (IS) organization. On the one hand, since the environments in which its customers operate are undergoing rapid discontinuous change, the IS organization needs to be able to rapidly adapt its customer's computer systems in response to that change. On the other, the IS organization itself operates in a changing world, inherited partly from its customers and partly from the broader environment. Thus it must organize itself and its own systems (such as CASE tools) for adaptation.

1.2. Inspiration: Change occurs in shearing layers

Truex, Baskerville and Klein [40] view an enterprise as an emergent organization[2] and recommend that information system developers embrace and even foster change within both the organization that they support and the information systems that they maintain and evolve. As such, they propose an alternative goal set[3] for information systems professionals that optimizes on high maintenance rather than low maintenance. In their words:

> Emergent IT organizations value continuous analysis, negotiated requirements, and a large portfolio of continuous maintenance activities.

But how can engineers optimize for change rather than stability, and do that efficiently, effectively and safely? Engineering wisdom seems to suggest that certain forms of change just cannot and should not occur quickly. Goals of high throughput and reliability, or a need to interface to a wide variety of other systems, are all believed to be incompatible with many forms of change.

Our source of inspiration for this dilemma is Stewart Brand's observation that successful buildings are constructed to support qualitatively different rates and scales of change [7]. That is, they are constructed in *shearing layers*, with a clear demarcation between parts that should change at different rates.

Brand describes six shearing layers of change in a building. (i) A site, especially when bordered by other buildings and roads in an urban setting, may constrain generations of buildings. (ii) A load-bearing structure may last for 30 to 300 years. (iii) The skin (exterior surfaces) typically changes every 20 years due to changes in fashion or technology or for wholesale repair. (iv) Services (heating, ventilation, wiring, elevators, communications, and so on) "wear out or obsolesce every 7 to 15 years." (v) Space plans in "turbulent commercial space can change every 3 years or so." And (vi) stuff ("chairs, desks, phones,") "twitch[es] around daily to monthly."

Brand's model corresponds closely to our own intuitions about the kinds of changes that are made to information systems. Some features change very slowly: credit card processing, or posting systems in banks. Others change very frequently, such as advertising features of web sites. Moreover, some engineering techniques and technologies exist to overcome cases where appropriate shearing was not initially supported. For example, the technique of screen scraping allows composite business functions and more modern user interfaces to be assembled on top of the unfashionable and obsolete character-based user interfaces of valuable, long-lasting systems built in earlier eras. Such techniques are the equivalent, in Brand's terms, of changing skin and space plan while leaving valued, reliable structure and site unchanged.

An area where we need to enrich Brand's work is that corresponding to the qualitatively different kinds and rates of change there is a correspondingly richly differentiated set of materials and practices for achieving those changes. For building, for example, different groups of experts lay foundations, produce structures from reinforced concrete, and design and assemble different kinds of facades [10]. In particular, the experts in selecting and arranging "stuff" are the "users" of the building. As a result, the materials and elements of "stuff" are optimized so that almost anyone can rearrange them for their own purposes whenever those purposes change.

1.3. Bringing a consideration of shearing layers into three aspects of design

We propose to use scales and rates of change, and a corresponding richly differentiated set of materials and practices, as primary criteria for the separation of concerns not only in information systems development, but for three related areas of design:

- the design of organizations: with an emphasis on design that promotes organizational adaptiveness and learning;
- the functional design of information systems: that is, design of how information systems will support the organization;[4]
- the implementation of information systems: that is, design of how required function is made available to users using enabling information technologies.

While this may seem like a broad focus for study, any large information systems services organization must not only cover this range of practices, but do so in an increasingly

seamless manner. Moreover, if it is itself to be a learning organization, it must apply these three forms of design to itself, not only to its customers.

1.4. This paper focuses on functional design of software

Other papers, for other audiences, will address the shearing layers model of information system design from other points of view, notably those of computer-supported cooperative work, management information systems, organization design, middleware technology, and programming language design.

This paper focuses on functional design. Functional design is the territory between experts in information system design and people with expertise in the business in which requirements are continually (re)negotiated.

We propose constructs called *agents* and *conversations* as means for reasoning about the functional design of adaptable software. The bridge to organization design comes from determining[5] which people should engage in what conversations to achieve what outcomes. The bridge to software implementation comes from determining how to deploy each conversation within available middleware and other technologies.

In the next section we present at a conceptual level our constructs for functional design, based on our early experience of their concrete realization in a prototype system called Ibex. A detailed presentation of the Ibex language and the issues involved in its design is beyond the scope of this paper. We assert that such a language can be designed and explore how it will support information system development in shearing layers.

In section 3 we discuss how an agent / conversation model serves as the basis for the deployment of function as implemented software. Central to this is an evaluation of alternative technologies in terms of their costs-to-deploy-and-to-evolve, which allows us to refine our criteria for drawing boundaries between conversations. In section 4 we identify and comment on further related work before presenting conclusions in section 5.

2. FUNCTIONAL DESIGN IN TERMS OF AGENTS AND CONVERSATIONS

Our approach to functional design is to present an informational view of an organization as a fairly large number of units called conversations. In addition to basic issues of what information each conversation should store, and how it can be modified and visualized, each conversation can be reasoned about from two quite different perspectives:

- cooperative work and organizational design: who may participate in the conversation? who must pay attention to it and under what circumstances? who should have what awareness of the conversation?
- implementation design, which relates to sets of similar or related conversations: from which devices will which conversations be accessed? what throughput is required? what are the expected rates and scales of change?

The result is a conceptual view of an information system as "lots of applications,"[6] each supporting a single conversation or an interaction between a few conversations.

2.1. Agents participating in different genres of conversation as a model for information systems

In choosing agents and conversations as primary constructs for the functional design of information systems, we are building upon the considerable body of sociological studies of organizational work conducted within fields such as computer-supported cooperative work.

Lucy Suchman applied the insights of Harold Garfinkel [19] to propose a view of human action as improvised within each given situation, and in so doing successfully disposed of then-prevailing procedural views of work [38].[7] While she did not deny that people did indeed produce and make use of plans and procedures, she explained that when they are produced they are used as "resources for situated action" rather than as accurate accounts of what has taken or should take place.

Our focus in designing both organizations and software is on carefully identifying and designing situations, and some of the resources available to people within those situ- ations. With this approach we leave people free (empow- ered) to choose how to act within those situations [20,35].

We define a conversation as a virtual space grouping virtual resources for use by a specified group of agents.

In using the term "virtual space" we are drawing an analogy between how people use physical space and how they structure their use of information technology. On the one hand, people gather resources into shared physical spaces. For example, a room may contain posters, writing on white boards, papers and books all pertaining to a software development project. Similarly, a virtual space supported by information technology groups virtual resources for some purpose. On the other hand, people partition physical space into realms with differing degrees of exposure or intimacy, varying from public to private.[8] As Clement and Wagner have pointed out, human communication consists of similarly fragmented spaces, in which choosing not to articulate something within a given space can be as valuable as articulating it [9].

Virtual resources include information and means (such as an information model) for structuring that information and maintaining its integrity (such as operations that maintain the invariant of the information model). They include user interfaces (forms and dialogs) that present the information in useful ways, and means for navigating to other related conversations. Although agents may include both people and computer programs, we definitely do not see people and software as interchangeable.

People (human agents) are assumed to have a considerable degree of freedom not only in what they do and how they act within any given conversation, but to which

conversation(s) they pay attention at any given moment. This leads to a number of important concerns, including: awareness of what others are doing within a space consisting of a large number of conversations (such as through Erickson *et al*'s notion of social translucence [15]); producing an interactive account of your own behavior (as do the ground traffic controllers described by Suchman [39]); and many related visualization issues.

The conversation construct is intended to support the full spectrums between computer- and socially-enforced conduct, and from high to low enforcement. Conversations may involve only human agents (for example, e-mail or a chat room), a mixture of human and software agents[9] (for example, a person managing their bank accounts online), or only software agents (for example, ensuring a correct transfer of funds between two bank accounts).

Any structure within a conversation may be partially enforced by the information technology and partially by social pressures. As an example of how successful social pressures can be, Erickson describes a spontaneously formed limerick telling conversation within a chat room in which strict yet slowly evolving rules were rigorously enforced purely by social conventions [14].

2.2. Elements of a single conversation

A set of agent and conversation specifications constitutes a functional design for information systems to support certain parts of an enterprise.

A conversation stores and structures information for a group of agents. It maintains the integrity of the information by only allowing changes through operations known to maintain a conversation invariant. It renders the information to users in meaningful "forms" (user interfaces), which also support the dialogs that enable human agents to invoke operations.

The information stored in a conversation is specified in terms of an information model which defines types, relationships and attributes.[10] Each informational thing represented in the conversation is characterized in terms of a changeable set of classifications (types), relationships and attributes. Invariants govern legal combinations of types, relationships and attribute values.

A conversation's operations and forms govern changes to and the display of its information. Operations are specified in terms of their parameters, a precondition, a postcondition[11] and a condition determining to whom the operation is available. For Ibex, we assume that all operations within conversations are of short duration and are serializable. Forms are specified in terms of display and dialog elements, layout information, bindings of dialog elements to operation specifications, and conditions determining who may use the form.

A number of variables and types are built into all Ibex conversations. Within each conversation there is a unique thing of type *ThisConversation* which represents the current conversation and is bound to the conversation variable *thisConversation*. Each person currently involved in the conversation is represented by a thing of type *CurrentlyActivePerson*. Operations and forms may refer to the session-level variable *currentUser* which identifies the *Person* that represents that user within the conversation.

An information model together with operation and form specifications constitutes a complete functional specification of a conversation. The set of agents who may participate in the conversation is determined entirely within the *availableTo* clauses of form and operation specifications.

2.3. Correspondences and transactions between several conversations

While conversations are intended to be somewhat self-contained information spaces, users frequently switch their attention between conversations. Equally, one conversation may involve the identification of, collection of information about, or transfer of information to and from other conversations.

Two low-level features of Ibex support such connections between conversations. Firstly, a conversation may be represented within another conversation as a *KnownConversation*. Secondly, a formal correspondence may be established and maintained between things in two conversations. Thing correspondence is achieved in terms of identity only: thing *a* in conversation *A* is asserted to correspond to thing *b* in conversation *B*, and will remain so until the assertion is revoked.[12]

*KnownConversation*s and thing correspondence enable a navigational structure that spans many conversations. Additional conventions can ensure, for example, the ability to locate all conversations of a given kind or allow a user to locate all conversations in which they participate or are expected to do something.

These features also support transactions and multi-conversation forms. In Ibex, transactions are a special kind of operation that refers to and alters information within two or more conversations. They can only occur between conversations that are known to each other and can exploit and create correspondences between the things in those conversations. In a similar way, a multi-conversation form is a special kind of form that displays information within and dialog between two or more conversations.

Multi-conversation forms and transactions are intended to be short lasting, atomic and stateless. When this is not the case an intermediating conversation will more accurately represent the true nature of the business. As a conversation it may be long lasting and have its own state.

2.4. Criteria for demarcating conversations include durations and rates of information accumulation

In addition to providing basic design constructs we are collecting heuristics that help determine whether a given functional design is a *good* outcome of a design activity. The

determination of appropriate heuristics in this area is a subject of open ended research and reflection.

The following sections draw an understanding of heuristics for demarcating conversations from sociology and engineering. Sociological and business views of work allow us to demarcate conversations by asking which agents should be engaged in which conversations. The engineering viewpoint brings an understanding of the different costs of implementing and changing various sets of conversations using the many different techniques and technologies available to software engineers.

Remaining within the realm of pure conversation leaves us with a narrower although promising set of criteria. In this realm, conversation can be considered from the literary point-of-view expounded by advocates of genre theory.[13] We propose as a design heuristic that conversations should be of pure genres. While articulating what might constitute a pure genre or a tolerable hybrid is a subject of ongoing research, we can illustrate this intuition in terms of a tentative definition. We assume that this definition can be extended to apply to inter-conversation transactions also.

A tentative expression of how to define a conversation genre is: the patterns and subjects both (i) of participation by the conversation's human and software participants and (ii) by which they access and update the informational content of the conversation. Genre purity occurs when the patterns are clean and regular.

In this view, the separation of concerns by interactional patterns only can be seen as akin to performing a spectral analysis of a signal: that is, using differences in duration, rates and frequencies of participation and information accumulation to identify subconversations that require qualitatively different software implementations.

2.5. Transforming concepts from computer-supported cooperative work into functional design constructs

Fortunately many general lessons drawn from ethnographic and genre studies of work are available for our reuse. Not only is the agent / conversation approach inspired by sociological accounts of organizational work, we are in the process of reexpressing as patterns of conversation many of the well known and documented strategies that people use in their computer-supported cooperative work.[14] The resulting pattern language, we believe, will allow us to enroll organization members and their intuitive organizational understanding as a more natural way of separating functional concerns.

For example, in his work on "ethnomethodology" Garfinkel emphasizes that much of a person's behavior consists of producing accounts of her own behavior [19]. In applying this precept in a study of ground traffic controllers at an airport, Suchman observed that they produced accounts for two quite distinct audiences: to coordinate their decision making with that of their colleagues, and to make their work visible to managers and regulators [39]. Corresponding to this intuition are patterns that regard some forms as supporting accounts, form being the general purpose Ibex construct for defining required user interfaces. Forms can then be discussed as documents whose completion achieves some sort of accounting within the organization, be it amongst colleagues who are cooperating to accomplish some larger outcome, between employees and their supervisors, or both at once.[15]

As well as CSCW intuitions that in certain situations bring a sociological and organizational meaning to certain elements of the Ibex language, there are intuitions that relate to the demarcation of conversations. Sociological accounts of work are full of discussions of boundaries and boundary practices. Star talks about artifacts specifically developed to support exchanges between different communities of practice [37]. Wenger talks about boundary practices including legitimate peripheral participation which we can interpret as specially demarcated conversations on the boundaries of a community [42]. Many authors talk about the advantages and disadvantages of keeping one group's work invisible to others (e.g., [31]). Clement and Wagner talk about the value of not articulating something [9]. And Bannon and Bødker build upon Strauss's notion of articulation work to point out that maintaining a "common information space" requires extra work on behalf of its participants [4].

2.6. Conversations as a foundation for shearing layers

So far we have introduced agents and conversations as ends in themselves with only passing reference to our motivation for introducing them: namely, as a foundation for discussing shearing layers.

Firstly, conversation provides a uniform means of representing a broad range of required function across the full spectrums between computer- and socially-enforced conduct, and from high to low enforcement. Moreover, they do this free of consideration of in which layer each piece of function might be implemented.

Secondly they match what we believe to be natural boundaries and practices within organizations than appr-oaches such as classical object-orientation. A conversation-orientation places boundaries where they belong, matching natural points of shearing within human organizations.

Finally, they promise to raise the level of discourse for IS design away from technology and towards how the IS supports the organization. Amongst other things, this allows us to enlist users and other subject matter experts in achieving a shearing layers separation of concerns.

3. IMPLEMENTATION DESIGN AS FUNCTION DEPLOYMENT TO TECHNOLOGIES DIFFERENTIATED BY COST

We started this paper by citing the opinion of Truex, Baskerville and Klein that information systems should be built to assume organizational instability and change and a

large portfolio of continuous maintenance activities. That is, information systems professionals should optimize their own practices and the systems that they build to assume high maintenance rather than low maintenance.

We then formulated the problem of this paper as being "how can engineers optimize for change rather than stability yet do that efficiently, effectively and safely?"

3.1. Deployment target variety leads to cost variety

The solution that we propose, of which functional design in terms of agents and conversations is one element, is a shearing layers approach to information systems development inspired by Brand's model for buildings.

Shearing layers provides a language and metaphor for discussing adaptive change within information systems development. As in buildings, changes are differentiated in terms of their associated costs, durations and disruption, with relatively many cheap, rapid, unobtrusive changes, and relatively few large, costly, long duration, disruptive projects. Moreover, we should think of software as being constructed as shearing layers with each layer deliberately having associated materials, practices and costs appropriate for its scales and rates of change.

We are led beyond a simplistic mapping of shearing layers to elements within a "layered box" software architecture diagram by the fact that *several* of Brand's layers (site, structure, space plan and even stuff) accomplish the *single* end of partitioning physical space into smaller units. Similarly, even for a single technological function such as the persistent storage of data there is a wide variety of rates and costs of implementation. The wide variety of information technologies (such as database management systems), whether proposed by academia or available on the market, are a response to the highly differentiated needs of different applications. That there is such a wide variety of needs and a corresponding variety of solutions seems to be natural in any technological market and an empirical validation of the shearing layers model.

In the context of system implementation, we use the term deployment (of function) to cover the broad range from the software equivalent of rearranging the stuff of a building (for example, a user spending twenty minutes implementing a spreadsheet) through to the equivalent of investing in new structure or site (for example, a multi-year, multi-person century project to reimplement a high throughput, high reliability system such as a transaction posting system in a financial institution).

In evaluations of the cost of deploying function, money to change, time to change and level of disruption must all be considered. Financial costs to change include: purchase, licensing and upgrade costs of raw software technologies; costs of employing, developing, retaining and renting the human skills necessary to deploy and maintain the technology in support of selected conversations; and costs associated with the purchase, leasing, maintenance and operation of supporting hardware, including bandwidth, storage media and processors. Deployment costs apply both to deployment of individual classes of conversation and of interactions between conversations that may have been deployed to different technologies.

Suppliers of materials, practices and human resources, whether aimed at buildings or information systems, seek to differentiate and diversify their offerings in terms of the differing needs of their users. So far as we are concerned, a key criterion for differentiating these technologies is in terms of their deployment costs. Moreover, deployment costs should be appropriate for the scale and rate of change of the shearing layer(s) at which the technology is targeted.

3.2. Criteria governing deployment

The appropriate timescale and cost for a change determines and is determined by the layer to which function is deployed. Deployment may vary from the implicit in the case where a user rearranges stuff in an existing deployment, to a need for a major system development effort.

Of course, there are many other characterizations that affect how a conversation should be deployed. For example, it may need to be deployed so that a certain participant may participate using one or more given access technologies. Access technologies may include: desktop workstations on company premises; a customer's home workstation; a mobile computing device disconnected from any network; or a telephone and its keypad. Software technologies may support deployment-related features such as the replication of a conversation (as in Lotus Notes).

3.3. Working in a construction site

What might the rich portfolio of ongoing maintenance activities that Truex, Baskerville and Klein recommend [40] feel like?

We have already indicated that such a portfolio will be richly differentiated, with relatively many cheap, short, low disruption changes and relatively few expensive, long, high disruption projects. This variety of maintenance projects should give the non-IT person the same sense of ongoing change as ongoing shearing changes to the buildings that they occupy and visit. They will be surrounded by varying scales and degrees of construction and rearrangement, including those that they accomplish themselves such as rearranging windows on their screens.

In more rapidly shearing layers where system function supports the most rapidly changing aspects of the business, there will be a great many ongoing change efforts as a consequence of what Truex *et al* call continuous analysis and requirements negotiation [40]. These will be performed by the users themselves and by information systems professionals who can implement the changes as and when needed, at low cost, and with little or no delay.

We are focusing our current language and tool design on these most rapidly changing layers for two reasons. Firstly,

these layers are not well supported by available tools. While some products exist that allow rapid assembly of simple applications they do not integrate well with function deployed in other layers.

Secondly, we are firm believers in in-situ design techniques, similar to those advocated by Alexander for buildings [3,35]. This is because, in Wenger's words:

> There is an inherent uncertainty between design and its realization in practice, since practice is not the result of design but rather a response to it. [42, p. 233]

This is the argument for participatory, user-centered design. From such a standpoint, techniques such as prototyping and extreme programming can be seen as designing and verifying within a relatively cheap layer function that will later be deployed to a more expensive layer.[16]

3.4. Shearing layers makes sense of and brings rigor to choices of how to implement

While there already exists a broad range of technologies and techniques aimed at decoupling things that change separately (e.g., [18]) or at implementing things with different costs, there have been few attempts to make sense of these efforts within a single conceptual framework.

The shearing layers intuition addresses this by focusing attention on two things. Firstly, suppliers of and investors in infrastructure require an appropriately richly differentiated set of deployment capabilities, including corresponding skills, hardware and software. Secondly, they need techniques for separating required function into pieces each of which can be separately deployed to a single appropriate shearing layer and thus targeted to appropriate deployment capabilities. Functional design in terms of agents and conversations achieves the necessary separation of concerns and so enables rigorous consideration of costs.

4. RELATED WORK

So far we have demonstrated how our work builds upon work in areas such as computer-supported cooperative work, organization design and management information systems and provides a framework for understanding and exploiting the many techniques and technologies that seek to make software more adaptable. In this section we would like to explicitly relate our work to several specific themes of software engineering research.

4.1. Reflection in the lightest-weight shearing layers

Numerous commercial products have been developed to be programmable by end users, perhaps starting with spreadsheets in the early 1980s. More recently, many software technologies have been aimed at making engineers more productive. For example, anecdotally the development of applications in Lotus Notes is significantly cheaper than writing custom code with similar function in a high-level programming language.

Our approach is somewhat related to Dourish's Prospero, in which he applied the principle of reflection in his explorations of flexibility for CSCW applications [11]. (Dourish builds upon Kiczales' concept of open implementation [26]). In the Ibex language we apply reflection in making such concepts as conversation, agent, transaction and form explicit both in the Ibex language itself and in the models built with that language.

4.2. Making sense of viewpoints

In their short overview of the software engineering topic of viewpoints in software development [17], Finkelstein and Sommerville provide the following definition:

> The study of viewpoints embraces the relations between views, between views and agents, and between agents.

In such a definition, different agents are assumed to have different views on a situation because they have different positions within the organization. Viewpoints then become an obstacle to be overcome through consistency checking techniques and approaches to inconsistency management (for example, [13,16]). Approaches such as that of Verlage go further promoting the positions of agents to a first class, explicit status as roles whether or not roles were previously defined within the modeled domain [41].

In our approach we focus on social agreement and conflict rather than personal viewpoint. Conversations and transactions are locally and socially negotiated agreements of how certain forms of interaction can take place. As such, they are constantly renegotiated outcomes of necessary debates between members of the organization and are best explicitly specified within such debates. In contrast, viewpoint-centric views merely capture subjective opinions divorced from the very dialectical inquiry in which learning about interactions can occur [8].

Conflict can lead to learning and so is valuable to organizations [8,21]. As such software engineering practices should not unduly seek to eliminate it in the name of reconciling viewpoints. Within the shearing layers model knowledge of the extent of conflict or agreement over a certain form of conversation or transaction is considered to be extremely valuable. It provides an indication of how stable any functional design to support the interaction will be and thus to which shearing layer the interaction should be deployed.

4.3. Relationship to multi-schema systems

It is interesting to relate our proposal to debates in the late 1980s and early 1990s over infrastructures for producing integrated software engineering environments.

Much of the architectural side of this debate is summarized by Schefstrom [33]. Schefstrom compared the data-centric model of technologies such as PCTE with a more federated, control-centric model of technologies such as the ESF Software Bus. While the PCTE object base

supported schema definition as a series of tool-specific schema definition sets, these schemata nevertheless had to be resolved if tools were to in any way share data, even within a single PCTE object base. The Eureka Software Factory, on the other hand, assumed that the key archi- tectural concern was to allow message exchange between tools, leaving choices of techniques and technologies for data storage to each individual tool developer.

Our current approach is sympathetic to both approaches, generally favoring federated architectures, but valuing multi-schema technologies for those cases where several classes of conversation need to be codeployed.

4.4. Relationship to transactional and workflow models

We believe that not only can we achieve sophisticated transactional models using simpler constructs, but that it is better to do this than invest in ever more elaborate technological means to support intergroup transactional conflicts. We prefer to leave these as issues to be resolved within the social realm as negotiated practices, where appropriate coordination practices can be constantly renegotiated and even improvised.

We have presented a model in which work and control flows are largely implicit, being confined to pre- and postconditions of operations and what we call transactions. While our multi-classification object model allows roles to be introduced as classifications of *Person* things, we do not believe that formal roles are necessarily a good way of designing all aspects of work.

We are not at all sympathetic to the use of procedural prescriptions of work or work flows. Emerging approaches for organization design focus on designs that promote organization learning. They involve a shift from an agenda of procedure (define means) and control (of adherence to means) to one of accountability (for outcomes) and empowerment (allowing individuals to choose means and negotiate outcomes) [20].

However, we are very sympathetic to Bardram [12] and Dourish [5]'s inversions of workflow. Suchman is careful to observe that while human work is improvised within the situation and not the execution of predefined plans, plans and procedures are nevertheless often resources for this situated action [38]. Bardram and Dourish's responses to Suchman have been to build systems in which workflow representations are incorporated to help in the visualization of ongoing work in a user-meaningful way while in no way dictating how work is actually accomplished.

5. CONCLUSIONS

Our starting motivation was that enterprises need to become more adaptive, and that an aspect of doing that is having adaptable computer systems. The challenge is then to optimize information system development for change (high maintenance) rather than stability (low maintenance).

Our response is to make explicit within software engineering the notion of shearing layers, and explore it as the principle that systems should be built to be adaptable in response to the qualitatively different scales and rates of change to which they will be subjected. This allows us to separate function that should legitimately change relatively slowly and at significant cost from that which should be changeable often, quickly and cheaply.

While we are seeking to apply this intuition to three kinds of design, in this paper we have focused on the software engineering concerns of functional design and the deployment of function. We have described a set of constructs that encourage this separation of concerns during functional design and its preservation or controlled removal during deployment. Constructs include agents and conversations and others addressing the related concerns of what goes on within a conversation and between it and other conversations.

We have assembled various heuristics and criteria for demarcating conversations and have thus suggested how a shearing layers separation of concerns can be accomplished in practice. Here we rely upon genre theory and on the many results within the field of computer-supported cooperative work.

We have emphasized that information systems organizations require a richly differentiated set of materials and practices corresponding to the need to implement system function that is changeable on different scales and at different rates. The Ibex system on which we are work- ing is intended to have a dual role in this respect. On the one hand it will be used to cheaply build in-situ mock-ups of system function that may ultimately be built in more costly technologies. On the other, we expect Ibex applica- tions to be suitable as applications in cheaper layers: that is, for actual use, albeit on a local scale with low throughput.

There are many outstanding research questions the majority of which are beyond software engineering. For example, how do shearing layers within an organization relate to those within its information technology? This is a nontrivial question since it involves systems built of quite different stuffs. Part of the answer is that conversations are part of practices which are themselves situated in different shearing layers of the organization.

While we do not wish to claim that shearing layers is a new phenomenon in information system development, it is a phenomenon that deserves to be named. It requires theory and practices that embrace it. This paper has demonstrated how we are going about doing just that.

Acknowledgments

Ibex is joint work with Bard Bloom. It is part of the larger Enterprise Builder project involving Darrell Reimer, Mark Wegman and Sarvamangala Jagadeesh. We have benefited greatly from our discussions with Steve Haeckel and Doug McDavid about Sense and Respond, Tom Erickson and John Thomas about social aspects of computing, Stan Sutton and Asit Dan about deployment to Middleware. Clay Williams,

Gene Hoffnagle, Steve Bello and David Bevington gave valuable comments on a draft of the paper.

End Notes

1 Here we build on rigorous definitions from systems science [1]. An organization may be purposefully adaptive because its parts — people and smaller organizations — are purposeful. A system is purposeful if it is ideal-seeking. Software, which is a mechanism, is at best goal-seeking (and so purposive), since it cannot itself change the ends that it pursues. Thus software is at best adaptable and if it is this is a property of the way that it is constructed rather than the way that it functions.

2 Truex *et al* contrast the conventional, outmoded view of the stable organization with what they call the emergent organization. They 'use the terms "emergent" and "emergence" rather than "emerging" because "emergent" refers to the state of being in continual process, never arriving but always in transition' [40]. Haeckel's notion of *adaptive enterprise* [20] goes further still, extending emergence-as-a-continual-process with an organizational strategy focused on and driving learning and change.

3 See opening quotation for a summary of a traditional goal set.

4 What we refer to as "functional design" is elsewhere referred to using a variety of terms, including functional specification, systems analysis and requirements capture. The term "design" indicates that deciding what function is required is a creative process. There simply isn't a set of requirements lying around waiting to be captured [3,35]. The term "functional" is used as a contrast to "implementation". Functional design, as we shall see, may involve prototyping-for-function (as opposed to prototyping-of-implementation) and other forms of mockup to facilitate the creative, learning processes of design.

5 We carefully used the term "determining" rather than "deciding." Organizations are the result of design actions that occur in shearing layers: on many scales and at different rates. They vary from relatively formal actions of executives to the kinds of improvised acts discussed by Suchman [38] and negotiated commitments advocated by Haeckel [20]. In the latter cases these design acts are merely aspects of other actions.

6 Thanks to Sarva Jagadeesh for this phrase.

7 Others have built upon Suchman's and Garfinkel's work to make extensive critiques of workflow-oriented models of work. These critiques apply both to system implementation in terms of workflow technologies and, perhaps more importantly, to the use of many workflow representations within information system specification. For a detailed example see Bowers, Button and Sharrock's contrast of how work flows "from within" versus how it is viewed or imposed "from without" [6].

8 A quick survey of Alexander's pattern language for the design of towns and buildings [2] reveals that a significant proportion of the patterns are related to achieving various levels of permanent or temporary intimacy or openness. That this is so is registered in the pattern Intimacy Gradient. More specific patterns supporting certain valuable forms of intimacy or gradient include Common Land, Public Outdoor Room, Couple's Realm, Children's Realm, Farmhouse Kitchen, Private Terrace on the Street, Alcoves, Window Places and Workplace Enclosure.

9 For the purposes of the current paper we are deliberately downplaying the notion of "software agent." On the one hand, there is a considerable and growing literature under the title of "software agents" which, since it refers primarily to implementation strategies, may ultimately prove useful in the realm of what we call "deployment." On the other hand, in Ibex we are exploring a program unit called a [software] agent, which is a scope grouping a set of transactions, multi-conversation forms and shared declarations. This definition draws an analogy between a person (a human agent) who participates, mediates between and switches attention between many conversations, and a role for a unit of software (a software agent) that similarly mediates between several conversations. This concept will be discussed in detail in forthcoming papers about Ibex.

10 For Ibex, our approach is based upon the ISO Reference Model for Open Distributed Processing (RM-ODP) including its General Relationship Model (GRM) [23,24,28].

11 In Ibex an operation outcome is expressed as a transformation.

12 The intuition that object identity might be the only useful point of correspondence between applications or viewpoints has been the subject of a long-lasting discussion between one of the authors and Bill Harrison.

13 Genre as a way of understanding and classifying organizational communications has received increasing interest within the fields of management information systems (for example [32]) and computer-supported cooperative work (for example [14,15]) since the publication of Yates' historical study of the evolution of forms of communications within business [43].

14 This is the subject of a forthcoming paper [36].

15 As Suchman observed [39], and others have observed in their own studies, many software systems conflate these two forms of accounting, with mixed results.

16 Our focus on tools for supporting the cheapest kinds of change explains our decision to base the Ibex language upon RM-ODP information modeling [23,24,28] rather than a classical object model. This reflects our personal experience from a variety of projects (e.g., [27,34]) that information modeling is simultaneously easier for business people and achieves specifications with greater precision than when we have applied the classical object model.

References

1 Russell L Ackoff, Fred E Emery. *On Purposeful Systems*. Aldine Atherton Inc., 1972.

2 Christopher W Alexander, Murray Silverstein, Sara Ishikawa with Shlomo Angel, Ingrid Fiksdahl-King. *A Pattern Language: Towns, Buildings, Construction*. Oxford. 1976.

3 Christopher W Alexander. *The Nature of Order*. Oxford. To Appear, 2000.

4 Liam Bannon, Susan Bødker. *Constructing Common Information Spaces*. In [22], 1997.

5 Jakob E Bardram. *Plans as situated actions: An activity theory approach to workflow systems*. In [22], 1997.

6 John Bowers, Graham Button, Wes Sharrock. *Workflow from within and without: Technology and cooperative work on the print industry shopfloor*. In [30], 1995.

7 Stewart Brand. *How Buildings Learn: What Happens After They're Built*. Viking Penguin, 1995.

8 C West Churchman. *The Design of Inquiring Systems*. Basic Books, 1971.

9 Andrew Clement, Ina Wagner. *Fragmented Exchange: Disarticulation and the need for Regionalized Communication Spaces*. In [30], 1995.

10 Howard Davis. *The Culture of Building*. Oxford. 1999.
11 Paul Dourish. *Using Metalevel Techniques in a Flexible Toolkit for CSCW Applications*. ACM Transactions on Computer-Human Interaction, 5(2), pp. 109-155, 1998.
12 Paul Dourish, Richard Bentley, Rachel Jones, Allan MacLean. *Getting Some Perspective: Using Process Descriptions to Index Document History*. In Stephen C Hayne editor, *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP '99)*. November 14-17 1999, ACM Press.
13 Steve Easterbrook, Anthony Finkelstein, Jeff Kramer, Bashar Nuseibeh. *Co-ordinating Distributed ViewPoints: the anatomy of a consistency check*. Concurrent Engineering and Applications. CERA Institute, 1994.
14 Tom Erickson. *Rhyme and Punishment: The Creation and Enforcement of Conventions in an On-Line Participatory Limerick Genre*. Proceedings of the Thirty Second Annual Hawai'i International Conference on Systems Science, January 1999.
15 Tom Erickson, David N Smith, Wendy A Kellogg, Mark R Laff, John T Richards and Erin Bradner. *Socially Translucent Systems: Social Proxies, Persistent Conversation, and the Design of 'Babble.'* In Human Factors in Computing Systems: The Proceedings of CHI '99. ACM Press, 1999.
16 A Finkelstein, D Gabbay, A Hunter, J Kramer, B Nuseibeh. *Inconsistency Handling in Multi-Perspective Specifications*. IEEE Transactions in Software Engineering. 20(8) August 1994, pp. 569-578.
17 Anthony Finkelstein, Ian Sommerville. *The Viewpoints FAQ*. Software Engineering Journal, 11(1), 1996, pp. 2-4.
18 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley. 1995.
19 Harold Garfinkel. *Studies in Ethnomethodology*. Prentice Hall, 1967.
20 Stephan H Haeckel. *Adaptive Enterprise: Creating and Leading Sense-and-Respond Organizations*. Harvard Business School Press, 1999.
21 Rudy A Hirschheim, Kalle Lyytinen, Heinz Klein. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge, 1995.
22 J Hughes, T Rodden, W Prinz, K Schmidt editors, *ECSCW '97: Proceedings of the 5th European CSCW Conference*. Kluwer, 1997.
23 ISO/IEC JTC1/SC21/WG7. *Open Distributed Processing — Reference Model: Part 2: Foundations*. ISO 10746-2 / ITU-T Recommendation X.902, February 1995.
24 ISO/IEC JTC1/SC21. *Information Technology — Open Systems Interconnection — Management Information Systems — Structure of Management Information Systems — Part 7: General Relationship Model*. ISO/IEC 10165-7, 1995.
25 Jane Jacobs. *Systems of Survival: A Dialogue on the Moral Foundations of Commerce and Politics*. Vintage, 1992.
26 Gregor Kiczales. *Beyond the Black Box: Open Implementation*. IEEE Software, 13(1), January 1996.
27 Haim Kilov, Helen Mogill, Ian Simmonds. *Invariants in the Trenches*. Chapter 6 of Haim Kilov, William Harvey editors, *Object-Oriented Behavioral Specifications*. Kluwer, 1996.
28 Haim Kilov, James Ross. *Information Modeling: An Object-Oriented Approach*. Prentice Hall, 1994.
29 Haim Kilov, Bernhard Rumpe, Ian Simmonds editors, *Behavioral Specifications of Businesses and Systems*. Kluwer, 1999.
30 Hans Marmolin, Yngve Sundblad, Kjeld Schmidt editors. *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work (ECSCW '95)*. Kluwer, 1995.
31 Bonnie A Nardi and Yrjo Engestrom editors. *A Web on the Wind: The Structure of Invisible Work*. Special Issue of Computer Supported Cooperative Work: The Journal of Collaborative Computing. 8(1-2), 1999.
32 Wanda Orlikowski, JoAnne Yates. *Genres of Organizational Communication: A Structurational Approach to Studying Communication and Media*. Academy of Management Review, 17, April 1992: 299-326.
33 Dick Schefstrom. *System Development Environments: Contemporary Concepts*. Part 1 of D Schefstrom, G van den Broek editors. *Tool Integration: Environments and Frameworks*. Wiley, 1993.
34 Mark Shafer. *Using Information Modeling to Define Business Requirements*. Chapter 15 of [29], 1999.
35 Ian Simmonds, David Ing. *A Layered Context Perspective on the Design of Information Systems*. Chapter 16 of [29], 1999.
36 Ian Simmonds, David Ing. *Integrating CSCW Intuitions into Information Systems Development by Means of a Pattern Language*. In preparation, 2000.
37 Susan Leigh Star. *The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Artificial Intelligence*. In M Huhns and L Gasser editors. *Distributed Artificial Intelligence 2*. Morgan Kauffmann, pp. 37-54.
38 Lucy A Suchman. *Plans and Situated Actions: The Problem of Human-Machine Interaction*. Cambridge, 1987.
39 Lucy A Suchman. *Technologies of Accountability: Of Lizards and Aeroplanes*. In Graham Button editor. *Technology in Working Order: Studies of work, interaction and technology*. Routledge, 1993.
40 Duane P Truex, Richard Baskerville, Heinz Klein, *Growing Systems in Emergent Organizations*, Communications of the ACM, August 1999, 42(8), pp. 117-123.
41 Martin Verlage. *About Views for Modeling Software Processes in a Role-Specific Manner*. In Laura Vidal, Anthony Finkelstein, George Spanoudakis, Alexander L Wolf editors, Joint Proceedings of the SIGSOFT '96 Workshops, ACM, 1996.
42 Etienne Wenger. *Communities of Practice: Learning, Meaning and Identity*. Cambridge, 1998.
43 JoAnne Yates. *Control through Communication: The Rise of System in American Management*. Johns Hopkins, 1989.