Technical and Social Features of Categorisation Schemes

Paul Dourish
Dept. Information and Computer Science
University of California Irvine
Irvine CA 92697
jpd@ics.uci.edu
http://www.ics.uci.edu/~jpd

INTRODUCTION

Categorisation is an inherent feature of much individual and collective work. Filing, coding, sorting, collating and comparing inevitably rely on schemes by which items can be categorised. Technical systems introduce a range of mechanisms for categorisation, from simple file systems to intricate database technologies, and these systems are deployed into settings in which categorisation work is done.

Since categorisation is such a common aspect of many different styles of work, it is not surprising that it should feature in reports of the encounters between technology and work practice, and indeed it has. For example, Gerson and Star (1986) discuss the importance of negotiating between different viewpoints and charactisations of work, Suchman (1994) discusses the way that categorisations inherently reflect limited points of view, while Bowker and Star (1999) explore the range of uses and concerns reflected in the ostensibly "objective and scientific" taxonomies of medical terms. These studies are sufficiently well known that there is no need to go into their details here. Collectively, they draw attention to the way that category schemes are not simply passive tools through which categorisation work takes place, but rather are the outcomes of practices of meaning-making, and, at the same time, the medium through which those practices are carried out. They are inevitably partial, fluid and subject to ongoing appropriation and revision to meet the needs of diverse individuals and working groups.

Implicitly or explicitly, these observations about the role of categories and categorisation in working practice tend to be formulated as critiques of the technical conception of categorisation, as embodied in information systems. However, if we see the mismatch between technical and social understandings of categorisation as a technical problem, then it follows that we need to seek technical solutions, and find ways to reorient our technical notion of categorisation, making it a more appropriate basis for categorisation practice. My goal here is to tease out some of the problematic features of technical conceptions of categorisation, and discuss some potential alternatives that have been explored in various areas of computer science. First, though, I will briefly recount a case study of categorisation in practice which reflects some of the problems that emerge.

EXAMPLE: DOCUMENT CATEGORISATION

Dourish et al. (1999) presents the design of an information management system that emerged from an ethnographic study of document practices conducted by colleagues at Xerox PARC (Trigg et al., 1999). One particular focus of this investigation was how members of the group categorised, filed and retrieved documents in a central repository that was the primary coordinating resource for projects with a duration of years. The ethnographic work documented the problems that members encountered in trying to apply a single, organization-wide formal category structure to the individual and varying needs of each project. In the face of these problems, groups and individuals would produce their own, customized forms of the category structure which enabled them to complete their work. However, these customized category structures interfered with the mutual intelligibility of the files, since the details of the coding scheme were available only to group members, but not to other more distant members of the organization.

In developing a technical solution to the problems of filing and retrieval in this situation, we needed to find a means to support technical categorisation that incorporated these elements of the group’s working practices. So, the system needed to support the gradual and continual evolution of category schemes, reflecting the group’s evolving understanding of the relationship between potential category structures and their own work. At the same time, it also had to support the multiple perspectives that individuals might take on the files, depending on their relationship to the project. Thirdly, it had to support the relationship between these different perspectives, so that files and filing structures would be mutually intelligible despite the different perspectives that different members might have.

Our solution was built on top of Presto, an architecture for fluid document spaces, based on document properties as an organising principle. This provided a basis for a much less rigid form of categorisation than traditional systems would support, and allowed a much looser link between data objects and their assigned categories. However, it still left us to incorporate many other features by hand, making it clear that, to accommodate working settings of this sort, we need to look for new technical approaches to categorisation.

TECHNICAL APPROACHES

The reason that accounts of categorisation in practice feature so commonly in CSCW studies is that social categorisation operates so differently from technical categorisation. Indeed, the mismatch between social and technical accounts of categorisation is often seen as the source of a range of interactional and practical problems. Where natural categorisational practices are inherently subjective, fluid, and tailored to the moment and to the task at hand, categorisation as a technical phenomenon is objective, absolute and static.

For the purposes of this position statement, I want to take the mismatch between technical and social accounts of categorisation as a technical problem. That is, I want to look for solutions not simply by proposing revisions of specific designs, but rather by exploring the basis on which our technical accounts of categorisation are built. In particular, I want to outline two specific approaches – subject-oriented programming and predicate classes – which have emerged in the context of object-oriented programming systems. Object-oriented programming (OOP) is an approach to the development of software systems which essentially combines elements of data modeling and programming language design into a single framework. In contrast to traditional imperative programming approaches, which models programs as sequences of instructions that operate over a separate store of data, OOP provides programmers with an alternative metaphor in which their programs are conceived of as a set of distinct elements or objects, each responsible for some piece of the system’s functionality, and encapsulating both data elements and the ability to operate over them. A program’s execution is modeled as the sequence of interactions between these objects as they pass data items back and forth, make requests of each other, and so forth. So, building a program in the object-oriented style involves not simply specifying sequences of instructions, but also building a category structure by which the various possible objects in a program can be represented and related. This makes OOP a natural arena in which to look at the issues of technical categorisation, since categorisation in its classical technical form is central to the enterprise.

Subject-Oriented Programming

As described above, the OOP style links executable code directly to the data over which it operates, creating an active data structure called an object. Objects communicate by passing messages back and forth. For instance, one object might send another the message “print” as a request that it print itself on a nearby printer. The critical feature of this arrangement is the separation between *messages*, which describe operations, and the *executable code*, which performs them. The idea here is that each object “knows” how it should be printed. The sender does not tell the recipient *how* to print itself; it merely sends the message “print,” and the receiver uses its own code to handle this operation. The result is that many different sorts of objects might understand the “print” message, but will respond to it in different ways. The object which receives the message is entirely responsible for “deciding” how to respond.

Subject-oriented programming (Harrison and Ossher, 1993) is an OOP approach that challenges the idea that each object is entirely responsible for how it responds to messages. The idea behind subject-oriented programming is that different objects might have different perspectives on a target object; they might require different sorts of responses to their messages. So, for example, the “print” message might, in fact, require different behaviours when sent by objects representing a printer, a graphical interface, or a plotter. In subject-oriented programming, the response to a message (either an invocation of behaviour or access to state information) is determined jointly by the sending object and the receiving object, allowing different objects to have different perspectives on the rest of the system.

How does subject-oriented programming address the problems of categories in interaction? Primarily, it introduces a mechanism that allows us to move beyond the typical absolute or objective character of technical categorisations, and replace it with a model in which categorisations depend on the different perspectives which might be taken on a data element. To different people, or different aspects of a system, or different tasks, a single entity might appear differently. We can dissociate an object’s category from its identity, allowing objects to play different roles at different times.

Predicate Classes

In traditional object-oriented programming, the most important thing about an object is what sort of object it is – the *class* of the object. When an object is created, it is created as an “instance” of a specific class. In almost all object-oriented programming systems, an object’s class remains the same throughout its lifetime. Each object belongs to exactly one class. While classes can be related via subclass relationships (e.g. “apple” is a “fruit”, and “fruit” is a “food”, so that any instance of “apple” is also a fruit and a food), an object remains associated with one specific class until it is deleted.

Predicate classes (Chambers, 1993) present a different model. When a predicate class is defined, the programmer specifies a “predicate” or set of entry conditions for the class. At any given moment, any object in the system that meets the entry conditions is regarded as an instance of the class, and any functionality associated with that class will apply to that object. For example, suppose the programmer of graphics application creates a regular class, called “rectangle”, for rectangular shapes, and then creates a predicate subclass of rectangle, called “square”, with the predicate “left_side_length = top_side_length”. Users create graphical objects such as rectangles; and whenever the rectangles happen to have adjacent sides of equal length, the extra functionality associated with squares will automatically apply to them. So, predicate classes is a mechanism whereby the association between an object and a class can be made to be dependent on the immediate circumstances or the internal state of the object, rather than being fixed for the lifetime of the object at the point where it is created.

So, how do predicate classes address the mismatch between technical and social conceptions of categorisation? Predicate classes reflect a much more fluid notion of categorisation than traditional technical models. They reflect the idea that categorisation is, first, dynamic rather than static, and, second, a matter of “fitness for purpose” rather than absolute definition.

CONCLUSIONS

The mismatch between technical and social conceptions of categorisation is well known. Arguably, this mismatch between, on hand, the fluid, partial and task-specific categorisations we use naturally and, on the other, the fixed and rigid category structures adopted by technical systems, is responsible for a range of interactional problems and troubles incorporating technology into working practice.

In this brief position paper, I have shown two approaches, drawn from research into object-oriented programming, that attempt to make technical categorisation less rigid and absolute. In doing so, they implicitly adopt some features of everyday natural classification; and so, they hold some promise for addressing the observed mismatch. This is not to suggest that adopting an approach based on subject-oriented programming, predicate classes or some combination of the two will eliminate the problems we see with technical categorisation systems and work practice. The technical systems will still require designers and users to pre-specify structure rather than letting it emerge naturally in the course of the work being carried out. However, these and related approaches none the less offer some potential opportunities to see how technical categorisations can be made more responsive to the ways in which natural patterns of categorisation emerge.

REFERENCES

Bowker, G. and Star, L. 1999. Sorting Things Out: Classification and its Consequences. Cambridge: MIT Press.

Chambers, C. 1993. Predicate Classes. Proc. European Conference on Object-Oriented Programming ECOOP'93. Berlin: Springer.

Dourish, P., Lamping, J., and Rodden, T. 1999. Building Bridges: Customisation and Mutual Intelligibility in Shared Category Management. Proc. ACM Conf. Supporting Group Work GROUP'99 (Phoenix, AZ). New York: ACM.

Gerson, E. and Star, L. 1986. Analyzing Due Process in the Workplace. ACM Trans. Office Information Systems, 4(3), 257–270.

Harrison, W and Ossher, H. 1993. Subject-Oriented Programming (A Critique of Pure Objects). Proc. ACM Conf. Object-Oriented Programming Languages, Systems and Applications OOPSLA'93 (Washington, DC). New York: ACM.

Suchman, L. 1994. Do Categories have Politics? The language/action perspective reconsidered. Computer-Supported Cooperative Work, 2(3), pp. 177-190, 1994

Trigg, R., Blomberg, J., and Suchman, L. 1999. Moving Document Collections Online: The Evolution of a Shared Repository. Proc. European Conf. Computer-Supported Cooperative Work ECSCW'99 (Copenhagen, Denmark). Dordrecht: Kluwer.