The M.A.D. Experience:

Multiperspective Application Development in evolutionary prototyping

Michael Christensen, Andy Crabtree, Christian Heide Damm, Klaus Marius Hansen, Ole Lehrmann Madsen, Pernille Marqvardsen, Preben Mogensen, Elmer Sandvad, Lennert Sloth, Michael Thomsen

Department of Computer Science, University of Aarhus, Building 540, Ny Munkegade, DK-8000 Aarhus C, Denmark.

{toby, andyc, damm, marius, olm, pernille, preben, ess, les, miksen}@daimi.aau.dk

Abstract. This paper describes experience obtained through a joint project between a university research group and a shipping company in developing a prototype for a global customer service system. The research group had no previous knowledge of the complex business of shipping, but succeeded in developing a prototype that more than fulfilled the expectations of the shipping company. A major reason for the success of the project is due to an experimental and multiperspective approach to designing for practice. Some of the lessons to be learned for object-orientation are (1) analysis is more than finding nouns and verbs, (2) design is more than filling in details in the object-oriented analysis model, and (3) implementation is more than translating design models into code. Implications for system development in general and object-orientation in particular consist in the preliminary respecification of the classical working order: analysis – design – implementation.

Keywords: Large-scale system development, multiperspective application development, cooperative design, ethnography, object-orientation, rapid prototyping, evolutionary prototyping, OO-tools, experience report.

1   Introduction

In late January 1997 a globally distributed shipping company contacted CIT, proposing a joint project for the purpose of developing a prototype of a Global Customer Service System (GCSS). The initial concept of GCSS emerged from within the company as a means of supporting the global implementation of business process improvements (BPI). A month later a team was assembled that had never worked together before and had no knowledge of shipping whatsoever. A project description

was made and ten weeks and 5 major iterations later, the prototype was appraised and approved by the company's highest executive body. Work is still on-going. This experience report tries to describe central success criteria and lessons learnt to date.

Partners in the project (known as the Dragon Project) were the company (which cannot be named for commercial reasons) and the DEVISE research group.

The project was funded by The Danish National Centre for IT-research (CIT, http://www.cit.dk). The company is a large world-wide provider of containerised transport solutions, employing some two and a half thousand people in over 250 offices in some 70 countries on six continents. Having a superior customer service in an increasingly competitive market as its key value, the company has and is investing substantially in IT support for proposed business solutions.

The DEVISE group was originally founded with the research goal of increasing productivity and quality in the development of large, complex systems, and now incorporates a large number of diverse competencies. Participants in the Dragon Project include, from the university: one project coordinator, one participatory designer, three full-time and three half-time object-oriented developers, one ethnographer and for a part of the project, one usability expert. Participants from the business include senior management, business representatives from the major continental regions, administrative staff, customer service personnel and members from a private consultancy company.

Although the essence of shipping in our context might be described as the actual transport of cargo in containers, our concern is with the provision and delivery of customer services, which the transport of containers relies upon. Work here includes the handling of interactions with customers in formulating prices for transport (quoting), in booking containers, in arranging inland haulage, in documenting ownership of cargo, in notifying the consignee of cargo's arrival, and so on.

From this all too brief description, the job of design might sound like a relatively straightforward task but for many reasons the challenge that GCSS has to meet is anything but simple. Currently, no global system or formal practice exists, yet GCSS should support coordination between globally distributed sites as well as support a customer service that, while streamlined (i.e. operating with less resources), is both effective and efficient and at the same time respects local needs in the various regions. Thus, the main design issue in many respects was to develop a prototype supporting these needs. Important issues in addressing these needs consisted in:

- Developing an approach for obtaining detailed knowledge of shipping in a short period of time; this knowledge had to be developed during the project due to the very short project period.
- Similarly, the initial requirements and formal specifications for the prototype from the business were vague and had to be developed during the project.
- The architecture of the prototype / system had to be flexible and allow for local customisations.

In order to satisfy these issues we thought it necessary to form a development group consisting of people with diverse but nevertheless complementary qualifications, who

would be able to utilise their own specialities and at the same time be able to work together in an extremely focused group. The focus being to develop a prototype to support customer service. The roles of the different team members may, somewhat simplistically, be described as follows:

- The ethnographer: focus on current practice within customer service and related areas.
- The participatory designer: focus on the design of future practice and technological support with users.
- The OO developers: develop the object model, and implementation.

In what follows we describe the contributions to the project from the perspectives of ethnography, cooperative design and object-orientation. Having explicated what we take to be the approach's main strength, namely its experimental and multiperspective character in designing for practice, the paper concludes by stating lessons that we take to be of value with particular regard to object-orientation. In so much as object-orientation played a major role as a technology for developing a model of shipping and for implementing the prototype, then a number of lessons for object-orientation were learned. Perhaps the most important ones are:

1. *Analysis is more than finding nouns and verbs*. We will argue that it is often necessary and useful to base analysis on much more powerful means than currently advocated, such as ethnographic analyses of the social organisation of work.
2. *Design is more than filling in details in the OO analysis model*. Here we will argue that often techniques from participatory or cooperative design are necessary and useful in formulating concrete design-solutions and relating them to practice.
3. *Implementation is more than translating design models into code*. Our experience shows that it is important to start implementation early in the design. The initial design model will always be changed during implementation and the feedback from initial implementation should appear early in the project so as to facilitate further development.

The above lessons are of course not relevant for all kinds of projects but for a project as the one described in this paper we think they are relevant. Perhaps most important are the implications our work has for structured techniques of object-oriented analysis, design and implementation. Specifically, towards the inadequacies of formal development methods in accomplishing rapid prototyping.

2 M.A.D – system development using a common frame of reference

The Dragon development team came from backgrounds ranging from ethnography to cooperative design to object-oriented development. These competencies should not be treated in strict separation but rather, in the ways in which they interact and complement one another and in this way achieve a synergetic effect.

A guiding principle in this multiperspective approach was (indeed is) an orientation to a common frame of reference: everyday business practice in customer service. In this paper, we use an example – a real world 'instance' of working practice, namely "rerouting" - that was employed by all the perspectives at work to show how design was practically achieved. In so much as the instance was oriented to by each perspective, we use it to illuminate the *character* of each perspective. Furthermore, in so much as the instance was employed by each perspective, then we use it to illuminate the *ways* in which the perspectives *interrelated* in practice and, reciprocally, while preserving individual achievements, to display the *organisation of work* from which GCSS emerged as a design product.

The instance of rerouting was only one of many used during development. Furthermore, the competencies or perspectives did not treat the instances sequentially but rather, treated them in parallel or at different times thereby mutually informing one another and the prototype as work progressed.

In order to display the organisation of multiperspective application development in rapid, evolutionary prototyping, we now turn to a description of how the three main perspectives, that is the ethnographic perspective, the cooperative design perspective and the object-oriented perspective, approached development and coordinated their activities into a coherent 'process' from which a concrete product (more than) satisfying the requirements of a large geographically distributed organisation emerged. After this account we will outline some of the major lessons learnt.

3 The ethnographic perspective

Whether operating within the context of rapid prototyping or not, the ethnographic perspective places an emphasis on securing a real world reference to work and organisation, informing the design of cooperative systems (Heath et al., 92; COMIC 2.2, 93; Blomberg et al., 94). Working on the presupposition that systems design is work design (in contrast to model building for example[1]), it might otherwise be said that ethnography seeks to base design on an 'understanding of praxis' in which the system is to be *put to work*. The intention is to predicate design on enacted practice thus supporting the development of systems that resonate or 'fit' with the activities in which they are to be embedded, thereby circumventing a major cause of systems

---

[1] This is not to abnegate modelling but rather, to remind designers of the relevance of modelling – it is not an end in itself but a means to an end, namely the (re)organisation of human activities.

failure (Grudin, 89; Shmidt et al., 93; Hughes et al., 94). The notion of understanding praxis is 'bracketed' in that it is selective – ethnography orients to praxis in a certain or particular way.

From ethnography's point of view, work is seen to be socially organised with organisation itself emerging as a local production out of the *routine accomplishment and coordination of tasks* experienced by practitioners (and generally understood) as the working division of labour (Anderson et al., 89). Ethnography's task is to make the working division of labour visible and available to designers in the concrete details of its accomplishment – i.e. in terms of work's real time organisation in contrast to idealised form (Rouncefield et al., 94). This is achieved through direct observation and naturalistic description of working practice portrayed from the point of view of parties to the work (Crabtree et al., 97). The notion of 'social organisation' refers to the conventional ways in which activities are accomplished and coordinated by *any and every* competent member engaged in the work. The notion of 'convention' refers, conjointly, to members' reoccurring actions and interactions, the artefacts used and the practical reasoning employed in doing the work. Work's social organisation is discovered through 'mapping the grammar' of the domain by empirical instance[2].

Simply and briefly put, the notion of 'mapping grammar' refers to *ordinary language* which is seen to embody 'language-games' (Wittgenstein, 68). Language-games are distinct practices - computer science, sociology, customer service in shipping for example are *unique* ways of talking and thus of *acting* in the world – each of which consists in a unique family of concepts ('quoting', 'export handling', 'documentation', 'rerouting' in customer service for example). In so much as language-game concepts *are* activities and in so much as they are *intersubjectively* employed ~ enacted by a unique *collectivity* of people (customer service staff for example), then to describe the actual *performative* details of their use is to describe the *social* organisation of the named activity. The activity of description is called 'mapping', the concepts mapped the 'grammar' of the language-game, the description itself the 'instance'. Empirical examples or 'real world' instances of language-game concepts-in-use not only preserve the social organisation of work in describing that organisation *in its own terms* but in so doing *make visible* what the work is 'really all about'[3] (Hughes et al., 92).

3.1   The Bremerhaven Instance (1)

Mapping grammar, a practical example: Customer service work in the container shipping business consists in the activities of 'quoting', 'export handling', 'alloca-

---

[2] The notion of 'mapping grammar' is explicated in *Talking Work: language-games, organisations and computer supported cooperative work* (Crabtree, forthcoming).

[3] It should be said that in so far as ethnography does preserve the real world character of work then it does so through an attention to concepts-in-use, to 'rerouting' as a contingent but nevertheless routine human achievement in, of and as work in contrast to 'rerouting' as an abstract information process (Garfinkel et al., 70; Hughes et.al, 96). This is not to rule abstraction out of play but to ground abstraction in formal properties of enacted (in contrast to idealised) praxis

tion', 'documentation' and 'inbound handling'. These concepts or categories delineate the working division of labour. One important feature of allocation work is to 'reroute' freight in response to contingencies. Rerouting occasions collaboration between equipment management, export and import handling, and documentation[4] and is itself occasioned for various reasons: bad weather, customer requirements, running off schedule etc. Rerouting consists in 'reallocating' containers to a substitute vessel or vessels - there may be several 'legs' and different vessels on any particular container's journey. The substitute vessel may be located in a different port to either the load or leg port. More: the destination port of rerouted cargo from a particular vessel may well be different. Thus, the activity of rerouting is all about arranging appropriate transport for cargo going to multiple destinations from some contingent point either to the destinations direct or, failing that, to a point from which cargo can be delivered to its respective destination ports. Real world instances of rerouting's accomplishment – actual *responses* to a vessel being delayed – revealed that route alterations occasioned not only changing the first leg of a journey but also the first half of the second leg for instance, or, indeed several legs on a journey. Furthermore, these changes were discovered to be subject to criteria of rerouting, specifically of time and cost: independent local carrier was specified wherever possible if time allowed, rail failing local carrier, truck failing rail; local carrier is more cost effective than rail, rail more cost effective than truck although time pressures may necessitate everything being moved by truck. It also transpired that rerouted cargo must be grouped: in some locations refrigerated containers, used for perishable products in particular, cannot be moved to transhipment points by rail for example (due to concerns of supervision – the temperature of refrigerated containers must be monitored at regular intervals), freight for this or that destination must be identified and the criteria applied. Furthermore, as a result of 'hard-wiring' working processes into the existing system, rerouting had to be accomplished individually – groups of freight could not be selected and assigned to alternate vessels except in the simplest of cases: 'roll over', i.e. temporal rescheduling to the next available vessel.

3.2 Analysing the instance

Mapping the socially organised details of activities accomplishment by empirical instance not only allows us to understand praxis in the context of the working division of labour[5] but in so doing, to identify *practical problems of work* and their *situated methods of solution*. Taken together, the practical problems of work and members' intersubjective or shared methods of solving those problems, and of solving them routinely, day-in-day-out in the face of any and all contingencies, show us just 'what

---

[4] Space prevents a detailed description of the work involved here but in practice (i.e. design) these details were central to the formulation of design-solutions.

[5] Which enables us to get 'hands on' the actual ways in which activities of work are *coordinated* and thus identify what is essential to successful work-oriented design – if activities cannot be coordinated, work simply cannot be achieved and we can add another systems failure to the list.

the work is really all about' and make *concrete possibilities* for support through design *visible and available to* design. Thus, the 'instance' circumscribes a **problem-space** for design. More: in illuminating the socially organised ways in which staff routinely go about solving practical problems of work, the instance circumscribes a **solution-space** for design.

The outcome of analysing the rerouting instance for example, a collaborative task performed by members from each of the perspectives at work, was the development of a flexible 'tree structure' supporting local decision-making and the coordination of a complex task by actively displaying all destinations for cargo on a delayed vessel and through the application of which one can specify any number of leg changes and different modes of transport for any group of selected containers, updating all members of the specified group in one go.

3.3   The limits of ethnography

Clearly there is a great deal more to observing praxis than meets the unaccustomed eye. In securing a real world reference preserving the context of work the ethnographic perspective provides concrete topics and resources for design. In so much as system design is work design, ethnography thus supports the development of systems that resonate or 'fit' with the activities of work in which they are embedded thereby supporting the local production of organisation in innovative social and technical ways (Kensing et al., 97). Virtues aside, ethnography is not without its limitations: it is one thing to identify a **problem-solution space**, another to identify design-solutions – a very practical problem resolved in practice through prototyping.

4   The cooperative design perspective

Cooperative design (also known as participatory design), as developed in Scandinavia over the last decades, stresses the importance of creative involvement of potential users (end-users, managers, affected parties, etc.) in design processes (Bjerknes et al.,87; Greenbaum et al., 91). It does so both from a "moral" perspective – users are the ones who have to live with the consequences of design – and from a practical perspective – they are competent practitioners understanding the practical problems of work, a capacity which enables them to assess and / or come up with alternatives to design. From this perspective, design is seen to be a cooperative activity involving not only end users but also other groups with very different but indispensable competencies. A primary means to the end of designing successful products is, accordingly, to bring these competencies together. This is often achieved in workshop like settings through the appliance of the different perspectives and competencies on a common and concrete issue, e.g. descriptions of current work, scenarios for future possibilities, mock-ups, and prototypes. The underlying assumptions and claims as well as a number of tools and techniques for the practical employment of cooperative

design may be found in (Greenbaum et al., 91; Grønbæk, 91; Grønbæk et al., 93; Grønbæk et al., In Press; Mogensen, 94)

Closely related to the cooperative design activities, a number of usability studies were conducted. In this context 'usability' is attempting to transcend a history of work "with one user in a lab thinking aloud" and move evaluation into the workplace itself. In this respect, it is a deliberate attempt to move away from a conception of users as human factors to a conception of users as human actors, and from the notion of finding 'problems' to a notion of the user as an active participant in design (Bannon, 91; Bødker, 91; Wiklund, 94). Despite struggling with an in-built cognitive bias (Grudin, 89; Bannon et al., 93; Twidale et al., 94) particular workplace studies supplemented with more traditional one-to-one lab type evaluations focusing on human computer interaction (HCI), provided detailed information relating to specific usage of the prototype and thus supplemented cooperative design activities.

Cooperative design has, by and large and up until now, been carried out in situations where the potential users constituted groups of manageable proportions. One of the major challenges for cooperative design in the Dragon Project is to resolve problems of scale: the organisation is distributed across more than 250 offices in 70 countries around the world and users may be found on many different levels in the organisation. How does one work with such dispersed groups? How does one find representative users? How does one deal with all the different (and frequently conflicting) perspectives and interests among not only different levels in the managerial hierarchy, but also culturally and regionally dispersed groups?

The strategy employed so far can be characterised as a mix of two approaches. On the one hand we have worked in various specific areas, regarding geographical locations, specific work domains, and corresponding functionality (e.g. rerouting, initially with the Aarhus site as primary point of reference). Embedding design in actual practice has helped to ensure in-depth knowledge regarding the issues being designed for at the given time. On the other hand, and at the same time, we have tried to maintain and elaborate "the big picture", both regarding the functionality GCSS eventually will provide as well as the regional aspects in order to prepare for, or at least not counteract, later developments. Both strategies have been approached in parallel and, naturally, both approaches have influenced one another.

To date, most of the concrete design activities involving developers and users have been carried out in four ways:

- Presentations of the prototype with subsequent comment / discussion sessions (all in all 100+ users from over 20 countries).
- Workshops (1-3 days) elaborating the details of current problems, current stage of the prototype, and various alternatives (all in all 25 users from around 10 different countries).
- Continuous workshops analysing and designing aspects in various versions of the prototype (6 users from 4 countries).
- A series of usability studies (8) with the business representatives attached to the project and with customer service staff in the local office in Aarhus (one to two users at a time).

An example of how the first two types of interaction have worked is a recent visit to Singapore: Four developers and the two business representatives working with us in Aarhus arrived in Singapore on Thursday. Friday morning we presented the prototype and its intended use to some 20 people from various positions in the Singapore office for around 3 hours. In the afternoon we all joined various people doing their usual work in the office. Saturday morning, we had a workshop with four people centred on export handling. Saturday afternoon, we discussed lessons learned so far and decided on changes to be implemented immediately, issues for redesign when we came home, issues that were out of scope, and features that would be 'nice to have'. Monday, we split up. The ethnographer focused on issues where we needed more information (allocation and pricing), observing work-in-progress and interviewing people in the office. The cooperative designer and three users discussed and elaborated details regarding booking in a 'hands on' session with the prototype. The two OO developers started to implement changes prompted by the presentation and observations of work which were agreed upon on Saturday. Tuesday morning, we presented the changes in the prototype for around 10 people and went into detail regarding the next issue on the agenda, allocation and documentation (including pricing). Tuesday afternoon, we went to Malaysia. Wednesday morning, we presented the prototype in the Malaysian office …

4.1   The Bremerhaven Instance (2)

An example of the continuous work 'back home' between developers and users is the Bremerhaven example. When rerouting came on to the agenda, company personnel gave us a brief introduction to the problem. The ethnographer went to the office in Aarhus and collected a series of examples or instances of actual reroutings dealt with at the office. The cooperative designer started to come up with ideas for supporting rerouting based on existing knowledge and experience. The next morning was spent in a continuous 'ping-pong' between various suggestions for supporting rerouting and the instances of actual reroutings coming out of the ethnography. After three to four iterations, an understanding of the problem as well as a first suggestion for the design emerged. Both were presented to and discussed with business representatives in the afternoon. Shown in Figure 1 is an account of the emergent understanding of the problem as well as the first design-solution (in real time, the example and the design was constantly produced and reproduced on paper sketches and white-boards).

You have 200 bookings out of Aarhus, going to Rotterdam via a local carrier. 100 of these bookings are transhipped on the intercontinental, *OC1*, headed for the Americas; another 100 are transhipped on to another intercontinental, *OC2*, headed for Asia.

Problem: the local carrier does not call Aarhus this week, we have to reroute or reschedule the 200 bookings.
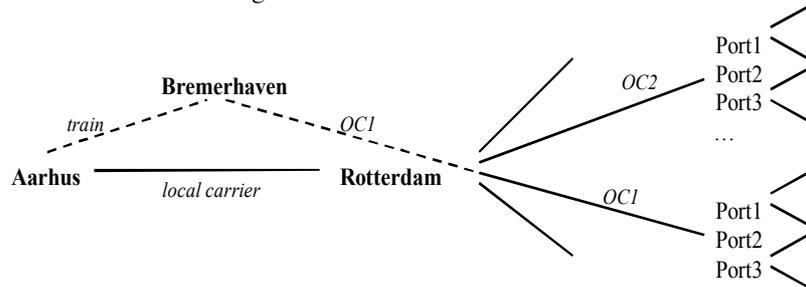
Solution: *OC1* calls Bremerhaven the day before it calls Rotterdam, so we can send all the containers to Bremerhaven by train, get them on the *OC1* a day before planned - everything is "back to normal". *OC2* does not serve Bremerhaven, we have to reschedule both for another local carrier and *OC2*.

Figure 1. The Bremerhaven instance

In considering how this is achieved in practice, it became apparent that affected cargoes' *destination* were different, spreading out globally like the branches on a tree. The suggested design was to represent all affected bookings in a tree structure with alternating ports and carriers. When the user, for example, clicks on *OC1* (see the above sketch) in the tree structure, it means that the user is now operating on all bookings out of Aarhus, transhipped in Rotterdam, and going out on *OC1*. Having selected the intercontinental carrier *OC1*, all further transhipment ports and final destinations may be seen. From here it is now possible to do rerouting via Bremerhaven for any particular group of bookings

Naturally, the above instance does not capture all possible problems in rerouting, neither does the design represent the final solution. The point is that the above understanding presented, in a very compact and understandable form, a good starting point and served as a powerful tool in further development where both the design solutions and the 'Bremerhaven instance' were refined and elaborated. As a paradigm case in point, the Bremerhaven instance was:

- Used and reproduced within the development group as common point of reference for design. The example states problems and solutions from current practice in respect to a specific design problem (designing for the rerouting and rescheduling of multiple bookings). Whenever we

encountered problems in the implementation, the instance worked as a common resource: whatever the specific design ideas and problems were, the quality criteria was always whether we could support the instance, not, for example, whether we fulfilled some predefined requirements.

- Used and reproduced in a large number of presentations and workshops between people from the development team and users from various locations and levels. It is an integral part of the prototype: on the one hand it explains a problem the prototype is trying to resolve, on the other hand, discussing ways of solving the problem triggers new understanding of the problem and possible methods of solution. Roughly a week after the first formulation of the instance, it was confronted with staff from a large 'import' port. As was pointed out, the design works very well for rerouting in out-bound, it does not work when you are sitting in in-bound, because here the focus is on what is coming towards you. The instance was expanded with that example, and in the design we catered for the option of having either the receipt or the delivery port as the 'root' in the tree structure.

- Used and reproduced in the usability studies where, embodied in scenarios, it provided the starting point and context for assessing the prototype. In testing the prototype, the instance as well as the design was further elaborated. Up until testing for example, we provided for either an 'outbound' or an 'inbound' view of the tree-structure. What came out of the usability studies in this respect was the idea that we actually needed both at the same time, facilitating an overview of both what was 'coming in' and what was 'going out'.

5   The object-oriented perspective

The OO perspective applied in this project is based on the Scandinavian tradition (Madsen et al., 93) where the focus is on modelling as opposed to technical concepts like encapsulation and inheritance. The first OO language, Simula was originally designed as a means for writing simulation programs and when writing simulation programs, it is useful to have a language with good modelling capabilities. The modelling approach has been very explicit in the design of BETA (which was the language used in the implementation of the prototype). One of the main advantages of object-orientation is that it provides an integrating perspective on analysis, design and implementation, but in order to fully realise this potential, there should be a proper balance and integration between modelling and implementation capabilities of the languages (Madsen, 96).

The application being developed (in this case the prototype) is considered as a physical model of a perspective on the problem domain. Selected phenomena and concepts in the application domain are synthesised into the model and represented as classes, objects, properties of objects and relations between objects. This model is a

very explicit part of the application and it can be regarded as a view on the application expressed in problem domain-specific terms. The latest version of the model is always the one in the application.

Below we discuss the concept of modelling, how the model was produced, the evolution of the model, the architecture in which the model is embodied, and the crucial role of tools in responding to amendment and change.

5.1 The concept of modelling

Working within our OO-perspective, system development is based on an understanding of the concepts used within the settings that the system will eventually support. The setting we refer to as the *referent system* and the process of translating referent system specific concepts to concepts within the computer system we refer to as *modelling*. The result of the modelling process we refer to as the *model system* or just the model (Madsen et al., 93, pp.289, Knudsen et al., 94, pp.52). Using the model within a referent system context we denote *interpretation*. This can, for example, be discussion of business concepts with business users while referring to the model system. Below we discuss how modelling was actually achieved.
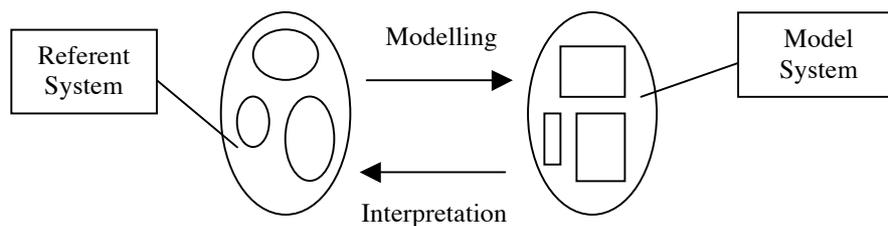

Figure 2. Customer service practice (referent system) in relation to the GCSS prototype (model system)

5.2 The production of the model

The activity of modelling was started right from the beginning of the project. Modelling has been an intentionally integrated part of each of the rapid prototyping cycles, in a sense on the same terms as the user interface and the architecture. The length of these cycles - the actual time spent on each – precludes a sharp distinction between the activities of analysis, design and implementation[6]. Working in an interdisciplinary context, our experiences extend existing notions of analysis, design and implementation. As such, a number of various resources and artefacts were used in producing the model. These included sessions with business representatives,

---

[6] A distinction which, as (Madsen et al., 93, pp 316) remind us, is highly questionable: 'In practice it is very difficult to do analysis without doing some design, and similarly doing design without doing some implementation.' Our experiences concur.

descriptions of the database of the current system, Yourdon diagrams of current and future business processes and, of the utmost importance, our own collaborative studies of *current practice*. The Yourdon diagrams, which came about as an output from the BPI analysis, exemplify the way in which the above resources and artefacts informed the development of the model, being used, much as use case diagrams may be used, as *'sensitising'* devices providing for the initial identification of objects. It should be stressed that the Yourdons were in no way used as requirements specifications (as such diagrams do not display actual practice in sufficient detail), rather it was a matter of using whatever resources were available to get an understanding of the problem domain.

Modelling proceeded in iterative phases: focussing on a particular area of business in collaboration with the business representatives attached to the project, the participatory designer and ethnographer; identifying important phenomena and concepts; analysing structure, and synthesising into a new iteration of the model. In the following example we exemplify and explain how modelling was achieved.

Modelling, a practical example - identifying concepts in the quote process*: By a combination of review constraints, formal business process descriptions stemming from the BPI and the need to start the modelling process with an accessible instance of business, the quote process was treated first. A particular discussion about the quote process with a business representative allowed us to identify important concepts and phenomena embodied *within* the quote process, such as 'corridor', 'routing', 'container', 'rate' and 'customer'. This session, as other sessions including other participants as mentioned above, was recorded in the form of a *blackboard snapshot* showing the concepts, some properties of the concepts, examples of values of properties along with textual and pictorial annotations clarifying or explaining referent system specific concepts. Figure 3 displays the features of the snapshot from the discussion with the business representative regarding the quote process.
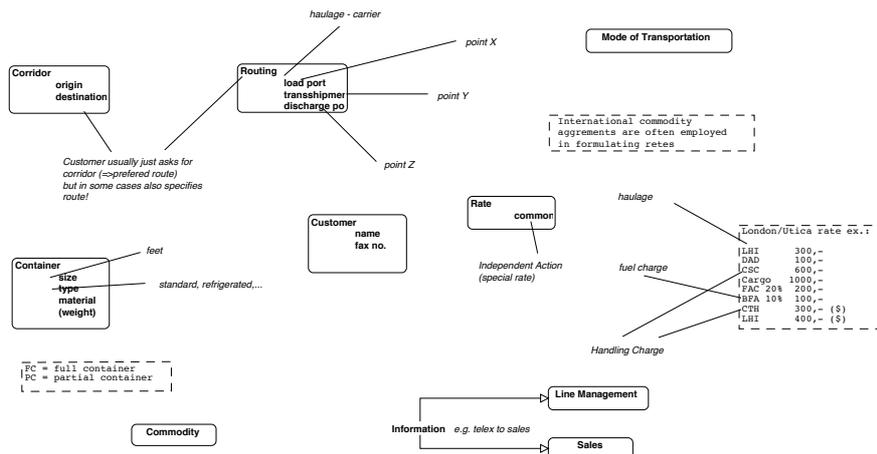

Figure 3. A blackboard snapshot

Blackboard snapshots produced through preliminary analysis of descriptions of working practice and the artefacts used were of considerable value and practical utility in creating the model. They provided rich detail and a firm basis upon which we were able to build a first model of the quote process. Furthermore, they worked as an important means of communication between the respective competencies at work.

Having created an initial model (see Figure 4), the model was then *interpreted* in the context of the referent system. The reason for this was to discuss the model with relevant members of the development team (including business representatives) in order to assess the state of the model and gain additional information as to the substance as well as the structure of the quote process. Other parts of the referent system were then analysed in the same way and synthesised with the parts of the model already elaborated[7].

Furthermore, the model was used for triggering *business* discussion. In this context the business representatives were able to see *referent system concepts instead of model system concepts;* comments on, for example, multiplicity's in associations, were put forward in the context and linguistic terms of business instead of in object-oriented terminology. In the context of formal reviews, in which participants from the
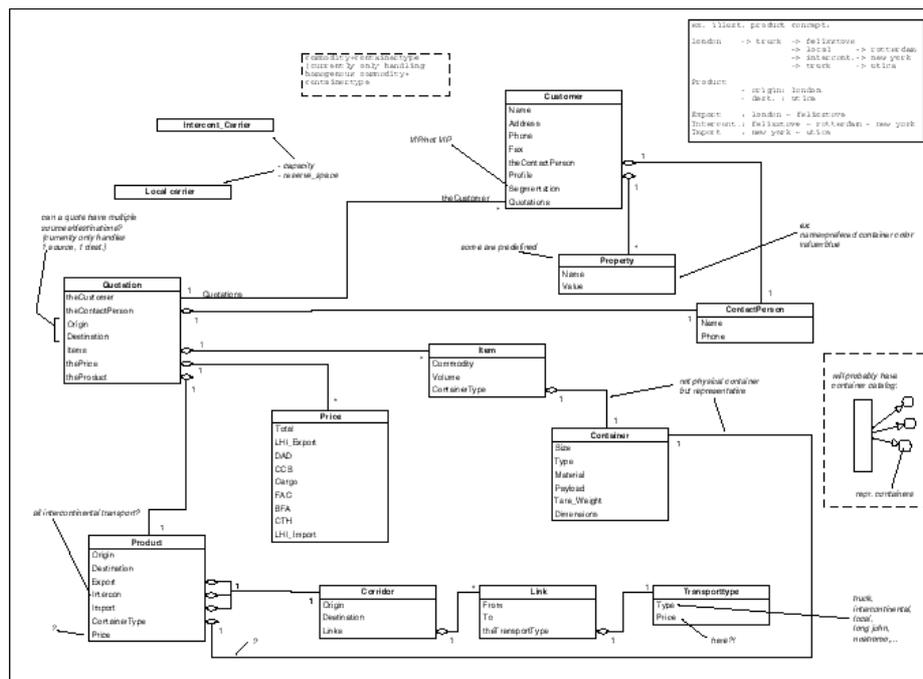

Figure 4. Initial design

[7] The success of 'snapshots' as a modelling technique has resulted in CIT investing in an electronic blackboard that we intend to use in future initial analysis / design sessions. This is just a first step in developing some supportive tools to support initial analysis / design.

business (including the consultants) had been involved in the BPI project and thus had an abstract perception of the business, the model triggered business discussions regarding *future practice*. It should be said that the model was not intelligible in all settings. In other contexts, participants who had not been part the BPI and who did not therefore have some knowledge of formal methodologies of design, had a hard time grasping the whole intent of discussing such a model.

As can be seen in Figure 4, the model exhibits technical simplicity: almost no implementation-specific attributes or classes have been added and only limited subset of UML's class diagram notation (Rational, 98), has been used namely inheritance, association and aggregation. In keeping the model "clean" it is kept interpretable and by using a formal notation the model is also executable in the sense that it is a living thing inside the actual prototype.

5.3   The evolution of the model

The model was by no means finished before coding started. It evolved along two dimensions: horizontally and vertically. Horizontally the model was gradually extended to cope with new areas in the referent system in the manner described above, with the first iteration of the prototype focusing on quoting, then booking and so on. In proceeding in this manner, both the model and the prototype were mutually elaborated. This was also the case vertically. The first iterations of a class or collection of related classes were focused on data and the relationships between the classes. In the following iterations the focus moved to high level functionality (the business functions), which introduced methods to the classes. Development of new business functions occasioned many amendments to the relevant parts of the model. Amendments primarily consisted in the addition of attributes and methods, modification of existing attributes and adjustments in class relationships.

Amendment to the model tended to be at a rather detailed level and our experience shows both that the model does not have to be complete either horizontally or vertically before coding can start, and that one does not need to spend too much time on the details in the design model before coding as they will be changed during implementation. The problem of when to start coding is not a question of validating the model but rather, in our experience, one of putting up and satisfying practical development requirements such as meeting deadlines for formal reviews. It should also be said that a major development requirement from our point of view is to have a running prototype even of minimal design to confront practice with and, in reaction, thereby elaborate modelling issues. Given this, our experience also shows that a model cannot be effectively created in isolation from implementation. As the model evolves together with the prototype in an evolutionary process, it becomes more and more stable with respect to the parts of the referent system so far covered. The Bremerhaven instance illustrates this.

5.4 The Bremerhaven Instance (3)

The problem of rerouting as elaborated through the perspectives and interaction of ethnography and participatory design and as visualised in the notion of a tree structure, emerged as a design issue relatively late in the process. Rerouting was on the agenda for the last review of the first phase of the project. At that time the basic functionality of the prototype had been developed to the point where the model supported a diverse set of business concepts including the concept of booking and related concepts such as products, schedules, allocation etc. The problem of rerouting is all about changing a bulk of bookings, a high-level operation that involves manipulation of multiple sets of bookings and related concepts. In so much as we already had the basic building blocks available, the development of the rerouting functionality was developed with only minor changes to the model. The basic structure did not need to be reconstructed. Radical reconstruction of the model was unnecessary because it was constructed on a natural understanding of the business concepts at work. Having said that, in dealing with the issue of rerouting, a new concept was introduced. The notion of rerouting being akin to a tree with an elaborate network of branches was not discovered among existing business concepts, but emerged out of a joint analysis of the work from the ethnographic and participatory design perspectives. Our approach to modelling allows new concepts to be introduced easily thus allowing the model to develop in an evolutionary manner.

The Bremerhaven instance is also a good example of how instances were used as a communication media between in this case the cooperative designer and the OO developers. Discussions between the cooperative designer and the cooperative developers regarding rerouting were centered around the Bremerhaven instance. The rerouting functionality is rather complex and it is difficult to get it right the first time. When problems arose during implementation the instance was used as a guidance for what at least should work in order to illustrate the basic idea. I.e. the Bremerhaven instance was used as a mediator in the design / implementation cycles as well as a minimal test case.

5.5 Prototype architecture

Working on the assumption that vertical, evolutionary prototyping needs to be performed within some well-defined architecture of the prototype, architecture was designed before actual programming on the prototype began, i.e. within two to three weeks. In this architecture the object model played a central role as being the *common frame of concepts* structuring the code in a referent system specific way. As we mentioned earlier, dealing with the concept of rerouting did not occasion any major reconstruction of the model, however, it did impact upon the architecture. Implementing rerouting in the prototype was problematic in that we did not know if the functionality required was already there, ready to reuse, or if abstractions of existing functionality were required. Through analysing the Bremerhaven instance we anticipated that major parts of the existing routing functionality could be reused. As it turned out however, the existing architecture did not provide the proper abstractions.

Constraints of time meant that the first design / implementation employed 'copy-paste reuse' of existing routing functionality. This was not construed as a problem in itself because, working in an evolutionary manner, we had the opportunity to design a new architecture in the next iteration cycle. As such, after the first major iteration of the prototype, we designed and implemented a component architecture (see Figure 5). This enabled us to implement rerouting using the cleaner abstractions afforded by the new architecture. These still, of course, need further iteration and as work continues more and more aspects of the Bremerhaven and other instances of work are brought into discussion. Change is now less problematic however, as the current architecture allows us to deal with the complex issues emerging from prototyping sessions: during the trip to Asia, for example, the power of component architecture was experienced as it was actually possible to add major components to the system "on the fly". The component architecture is the first step in the direction of an architecture with COM objects, (Microsoft, 95).
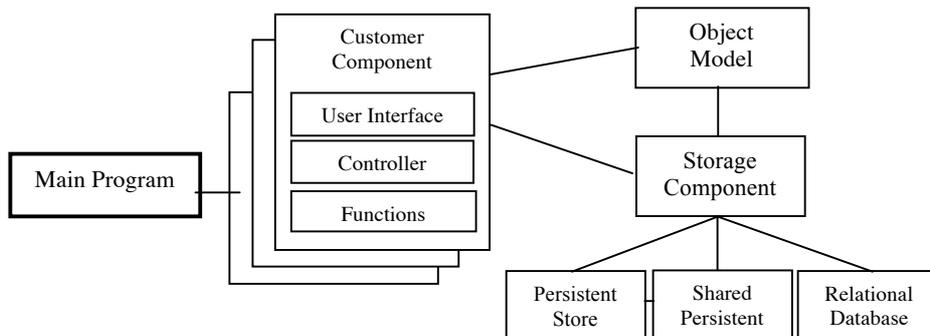

Figure 5. Current architecture of the prototype

5.6   The user interface of the prototype

In the user interface of the prototype we are combining an object-oriented user interface (in so much as part of the model is visualised in the user interface) with an activity-oriented user interface. The screen dump in Figure 6 shows this: business processes are displayed in a sequence of *process buttons* using a simple colour scheme for showing work's state of completion but not enforcing the accomplishment of work in a sequential manner, whereas the *tabbed controls* visualise the actual objects - customer, quote, transport - that are being worked on. Most upper tabs correspond to important domains in the model, whereas the sub-tabs visualise aggregated or referenced concepts.
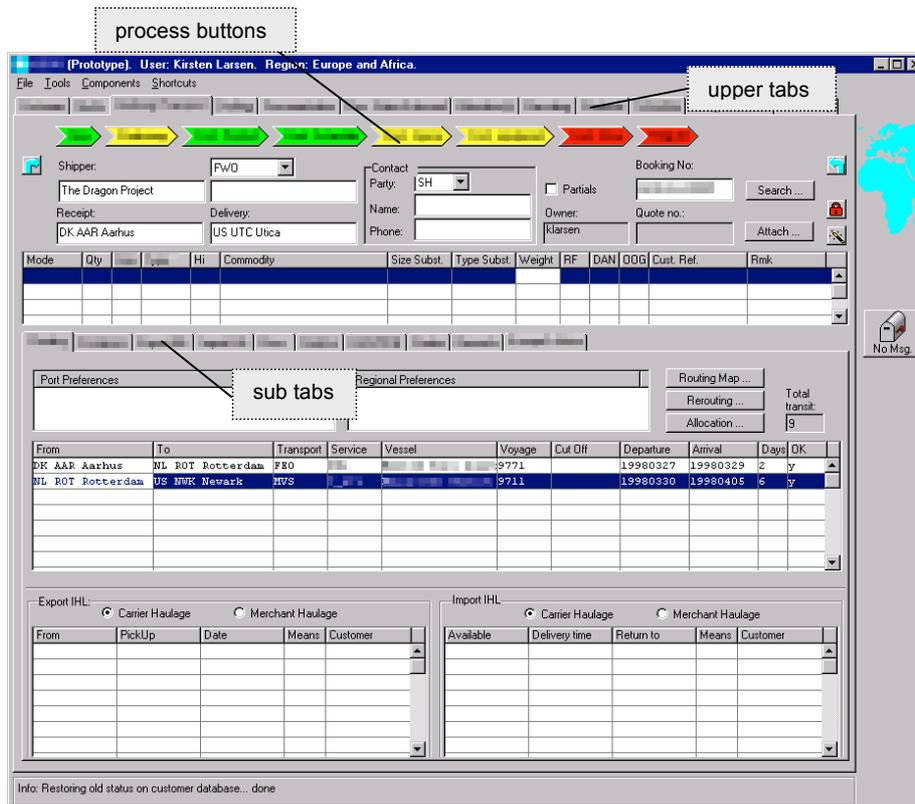
Figure 6. An overview of the main principles in the user interface of the prototype. (Image blurred for confidentiality)

6   Usage of tools

Within the context of any perspective the use of and support by tools is important. Working with evolutionary prototyping of architecture, functionality, user interface and model, tools supporting many iterations over a short period of time are a necessity. In the Dragon Project the main software engineering tools used are a CASE tool, a GUI builder, a code editor, a persistent store and a concurrent engineering tool.

The first four tools are part of the Mjølner System (Knudsen et al. 94, Ch. 2;http://www.mjolner.com) and the fifth tool used was concurrent versioning system (CVS, 98). The programming language used is BETA (Madsen et al., 93).

6.1 CASE tool

The CASE tool was used to draw the model using UML notation and to automatically generate the corresponding BETA code. In the beginning the CASE tool was also used to create the unstructured blackboard snapshots that later were used as input for the construction of the model. As diagrams of the model were created or modified, code was generated incrementally. Although code was thus available at any time, the early versions of the model were only used in the form of UML diagrams, as the emphasis was on communication and discussion.

The model can be changed via the diagrams or via the textual code. As emphasis moved to code and redesign cycles, model changes were mostly made in the code editor (see below) – only structural changes requiring overview of the model were made in the CASE tool. When the main emphasis is on coding, it is often more convenient to do the changes in the textual representation, especially when the changes are at a detailed level. Using the reengineering capability, the UML diagram was recreated from time to time and posters of it were made for discussion as mentioned earlier The reverse engineering capability is not based on having any additional information besides the BETA code, e.g. comments or other kinds of annotations, which means that the code editor – or in fact *any* editor – could be used to manipulate the model in its code representation.

6.2 GUI builder

The GUI builder is used to create the user interface in a direct manipulation graphical editor. Like the model the user interface can be changed via the graphical representation or via the textual code. The user interface was created from the start with the graphical editor, but was initially used for discussions only, although code is generated automatically. Changes regarding the physical appearance of the user interface, i.e. what type of UI controls used, and their concrete layout, were typically made in the graphical editor and changes regarding the basic functionality of the user interface and the interface to the model and functionality layer were typically made in the code editor. Due to reverse engineering and incrementality it is easy to alternate between using the two tools.

The user interface was often used as means for organising and/or coordinating activities between the cooperative designer and the OO developers. The initial user interface design was, by and large, made by the cooperative designer and it was then further elaborated by the OO developers in collaboration with the cooperative designer. Again due to reverse engineering it was possible for the cooperative designer to make changes to the user interface throughout the process, even very late in a prototype cycle.

The GUI builder generates code for a platform independent framework. This framework, which also had great importance to the development of the prototype, had to be extended during development in order to support platform specific UI controls. The new UI controls were manually inserted in the code, but they could coexist with

the automatically generated UI controls without affecting the reverse engineering process.

One problem or annoyance was that a large amount of time was spent on coding a strict programming interface to the user interface to achieve independence between the user interface, the model and functionality layer. The separation, however, showed to provide advantages in stability of interface and those advantages are considered to outweigh the difficulties. Furthermore, preliminary investigations show that much of this work can actually be automated (Damm et al., 97).

6.3 Code Editor

All of the tools mentioned above integrate with a structure editor. The editor knows the grammar for the BETA language allowing it to incrementally "catch" syntax errors and provide facilities for syntactic and semantic browsing of code.

Furthermore, the editor offers an abstract representation of the code, where any part of the code can be hidden. As became evident when a new developer was introduced to the system (and its architecture) relatively late in the process, this presentation of code and the semantic browsing facilities helped in gaining a quick understanding of the architecture of the relatively large prototype. Also, the abstract presentation of code and the editing operations at that level were helpful in restructuring even large pieces of code (Knudsen et al., 94, chapter 23).

6.4 Persistence

Another major player on the stage of tools was the persistent store that supports orthogonal and transparent persistence. In this way any object can be stored and given a *persistent root* all reachable objects will be made persistent transparently. This meant that only a little time had to be used on persistence issues in the first versions of the prototype. Persistent stores generated from data from existing databases simulated interfaces to legacy systems, and data non-existing in legacy systems was added. A point has been made of identifying those chunks that are within the object model to prepare for current work on interfacing to actual legacy systems. In the current version of the prototype the focus has turned to multi-user functionality in a client / server architecture and a storage component has been developed. The storage component (see Figure 4) constitutes a transparent interface to three different storage media: the initial single-user persistent store, a shared persistent store (Brandt, 93,94; Wiederhold, 97), and a relational database (for a selected part of the data).

6.5 Concurrent engineering tool

Although preliminary implementation was worked out on separate pieces of code the need for use of the versioning system CVS very quickly emerged. As implementation progressed it became evident that its presence was absolutely crucial: It facilitated

seven developers concurrently working on over 300 files containing over 100,000 lines of code, merging code with only minor problems. CVS was also used to merge changes made in regional prototype sessions with changes made at home at the University.

7   Managing development

Keeping the project "on track" and within the agreed time-frame was a major problem to be tackled. Project management and control was achieved in a number of ways. Development activities were managed and coordinated with BPI objectives through frequent business reviews ranging from the informal to the company's highest executive body. Up until now five different types of review, which vary both with respect to participants and with respect to purpose, have been held. A review of some kind has, on average, been held every two weeks during the five months of effective prototype development. In the second part of the project a major review of the prototype has been held approximately every second month. All reviews have actively involved relevant members of the development team. The reviews can be categorised as follows:

- Formal reviews with the project manager from the business, technical advisors from a consultancy and business representatives.
- Informal reviews with the same group.
- Reviews with the company's Regional Programme Coordinators (RPCs).
- Review with the company's executive body.
- Review with members of the company's Business Reference Group (BRG).

The purpose of the formal reviews – which were held more frequently in the beginning of the project than later on in development – was to assess whether or not the prototype was on the right track both in terms of scope and competency[8]. Informal reviews occurred in between the formal ones. They were informal in the sense that no single version of the prototype was built, frozen and presented for the occasion, rather it was a matter of the company's project manager and consultants visiting the development site and discussing work-in-progress[9]. Reviews with regional program-

---

[8] It might be noted that the "problem of culture" – academics working in a commercial context – was considered to be an issue of "high risk" by the company and its consultants; a problem resolved through frequent reviews.

[9] It should be said that for the first phase of development, the development team was located in company offices at a customer service site. While not actually located with customer service staff, access to practice was greatly facilitated. Furthermore, the transmission of knowledge was greatly facilitated through locating all the developers / perspectives in one room. This co-location supported internal awareness and coordination of development activities. It is an

me coordinators and the ethnographer, who did 'quick and dirty' studies of work (Hughes et al., 94) in Hong Kong and the US, were held to ensure that prototype kept a "global perspective" and, in their reaction to the current prototype, to get further input for development.

There has only been one review involving the company's executive body, the highest authority within the company regarding business development decisions. Following the executive review of the first major iteration of the prototype (May '97), the company decided to go ahead with developing a production version of GCSS. This decision was, amongst others, made on the basis of formal presentation of the prototype. The Business Reference Group, assembled in response to development phase two, represents the regions and is responsible for accepting features of the prototype that are to go into the production version and organising further, more "specialised" input from regional staff. By "specialised" staff is meant representatives from specific areas of the business. Although changed from time to time, one or two representatives from business have been located with the development team more or less constantly throughout design. They have been used intensively as resources on the problem domain and have also acted as coordinators between system development and BPI.

8   Concluding remarks

Returning to the question posed in the introduction, how come that the project was, and still is, successful? Based on the state of the work – still in progress – we will restrain ourselves from too firm conclusions. However, we still believe we might extract some lessons from the project so far. In the following, and we are very well aware that we are talking on the basis of one albeit comprehensive example, we will try to summarise some of the key findings in explicating some of the underlying means by which the prototype, the process, and the business results were achieved. Tentatively, and again rather simplistically, we can state that the main result seems to be the strength of an approach we might call an experimental and multiperspective approach to designing for practice.

8.1   Practice

Probably the most fundamental principle within our approach is the focus on the work practice the application has to support. One could say that practice is the subject of the analysis, a springboard for design, and the goal of the implementation. The business objective of the project as such is to design a product that fits the practice in which it is to be embedded and used, in this case customer *service*. Thus, the key feature of our approach is the focus on practice which is seen and treated as *the* fundamental resource in that it provides the possibility for grounding design,

organisation of work we have maintained in phase two although we are now located at the university.

identifying solutions to substantive problems, as well as providing triggers for new ideas as to how work (e.g. the *delivery* of customer service) might be achieved in the future through technological intervention.

8.2   Experimentation

In so much as design is an intervening activity very much concerned with future practice, one of the key characteristics of the whole development process is the quite comprehensive use of experimentation – i.e. the *performance* of analysis, design and implementation *in active collaboration with users*. Various methodological advise found in textbooks suggest that analysis, design and implementation should be carried out in sequence, although with iterations. The approach conducted within the Dragon Project can almost be said to go to the other extreme – there is no 'sequence' of work in the conventional sense of the word[10]. Schematically, Figure 7 depicts the approach employed and characterises experimentation.
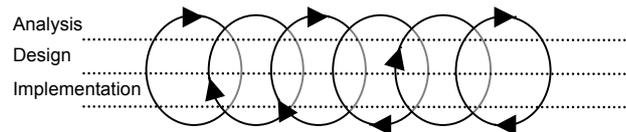

Figure 7.

In the analysis, we made extensive use of artefacts created within design and implementation. Similarly, the design activities were heavily dependent on both an in depth understanding of current practice (analysis) and a firm notion of what could actually be achieved within given constraints (implementation). Needless to say, implementation was dependent on analysis and design – notions all of which in this context are continuous, mutually elaborating and on-going: just as one starts analysing, designing and implementing in conjunction, then so one proceeds until a concrete product emerges

   This 'radical parallelism' and experimentation in all phases is important and, in placing users at the centre of design activities, leads to the implementation of a prototype suitable for continuation into the product system. Just as we designed using specific experiences still keeping "the big picture" in focus, we also, and simultaneously, tried to cater for a rapid prototyping process as we developed with the

---

[10] Of course, in so much as this approach is iterative and in real time demands restructuring of the product, implementation of architectures supporting integration with databases, existing systems and so on, then there is a 'sequence' at work but here we are talking about the development of a technological infrastructure in contrast to the development of a functioning system that the infrastructure serves. It is in the sense of the *process of developing of functionality* (i.e. what activities the system should support) that the notion of sequence becomes redundant as analysis, design and implementation are continuous, mutually elaborative and on-going in contrast to discrete step-by-step tasks.

product system in mind – i.e. we took and take an evolutionary approach to proto-
typing. Central to this process were the model and the architecture, the development
of which naturally requires a collection of suitable tools and techniques.

8.3 Multiperspective

In many respects the defining feature of the process is its interdisciplinary or
multiperspective character. As outlined in the previous sections we have made
extensive use of three very different perspectives. Although different, they share a
common frame of reference – the prototype and the practice it is intended to support.
Generally speaking we can say that:

- Ethnography provides a concrete understanding of work's real time
  accomplishment in contrast to idealisations and formal glosses.
- Cooperative design provides an understanding of the relationship
  between current and future practice through the experimental formu-
  lation of concrete design visions and solutions.
- OO provides a concrete relationship between design visions and the
  application in and through formulating a model utilising concepts
  derived from practice.
- The instances of work and the prototype provide and maintain important
  common reference points between the three perspectives throughout
  development.

Naturally, these perspectives do not have fixed boundaries; for all practical
purposes neither seeks to exclude the other. We all need to understand the practice in
question, we all need to share the design visions, as we all need to know what is
realistic in terms of implementation. On the other hand, for practical purposes, we
cannot all comprehend practice to the same extent. Thus, we made extensive use of
overlapping competencies and foci as well as overlapping activities and respon-
sibilities.

8.4 Implications for object-orientation

Besides these rather general lessons, we address more specific object-oriented issues.
Below is a summary of some of the notions and principles applied in the current
project ordered according to where they might best inform object-oriented analysis,
design and implementation.

OO analysis is more than finding nouns and verb.

- Analysis is in significant part directed towards understanding current practice.
- Ethnography is a powerful approach to understanding the social organisation of activities which *is* current practice.
- Developers need concrete experiences from within practice, complementary and in addition to bringing users into the development process.
- Prototypes, mock-ups and scenarios, complement ethnographic techniques in functioning as triggers for discussions on current practice with users.

OO design is more than filling in details in the OO analysis model.

- Design is seen as an on-going process of formulating "best matches" between current work and future possibilities.
- Cooperative design bridges between current and future practice by active user involvement in a creative process of experimentation.
- Concrete representations of design visions (prototypes, mock-ups, and scenarios) provide the possibility (1) for simulating future work through hands-on-experience and (2) for (thereby) formulating concrete design-solutions.
- The central 'challenge' in design is not so much to find representative users, rather it is to find users that can challenge representations.

And to complete the list:
OO implementation is more than translating design models into code.

- Implementation is also seen as the process of realising emergent, in contrast to predefined, design visions.
- Without understanding design visions and their concrete relationship to practice, it is virtually impossible to implement or find alternatives to formal specifications.
- Implementation is in significant part constructing primary *means* for analysis and design in accomplishing evolutionary prototyping.
- The construction of robust yet readily adaptable models and architectures are crucial in implementation and depend on flexible tool support.

Naturally, the above principles do not apply to all problems in all situations. System development is, afterall, a heterogeneous enterprise not only in terms of staff but also in terms of problem domains. Up until now each of the individual perspectives outlined here have been successfully applied to a wide variety of application developments in a multiplicity of situations. Although respective

disciplinary achievements suggested the strong possibility of developing a highly effective and unified approach to system development, in so much as this is the first major attempt at combining them in large-scale development, then success in a multiplicity of settings cannot be claimed for their particular association. Thus, it is difficult to assess scope, applicability, cost, etc.

What we can say, based on the experiences so far, is that the approach has been successfully applied in a situation characterised by the following features: complex human work practices, high uncertainty regarding the specifics of the potential application, large and geographically distributed organisation. In respecifying the classical working order of design from a sequential process of analysis – design – implementation to an on-going, mutually elaborative process dependent on active user involvement (experimentation), the multiperspective approach outlined here is potentially strong, both in terms of projected cost benefit and in actual terms of practical efficacy from a client's point of view, in supporting the integration of emerging information technologies into the world of work and organisation. To reiterate: we outline here an organised approach to, *not a formal method of*, work-oriented design. Finally, it might be said that in so much as we have explicated the acronym M.A.D. in what we hope is some reasonable detail, then that acronym not only captures the essence of a unique approach but also the frenetic character of rapid prototyping at work.

9   References

(Anderson et al., 89). Anderson, R.J., Hughes, J.A., Sharrock, W.W. (1989) *Working for Profit: The Social Organisation of Calculation in an Entrepreneurial Firm*, Aldershot: Avebury.

(Bannon, 91) Bannon, L.J. (1991) *From Human Factors to Human Actors: the role of psychology and human-computer interaction studies in system design*, Design at Work: Cooperative Design of Computer Systems (eds. Greenbaum, J. & Kyng, M.), pp. 25 – 44, New Jersey: Lawrence Erlbaum Associates.

(Bannon et al., 93) Bannon, L.J. & Hughes, J.A. (1993) *The Context of CSCW*, Developing CSCW Systems: Design Concepts, Report of COST 14, 'CoTech' Working Group 4 (1991-92), pp 9 – 36.

(Bjerknes et al., 87). Bjerknes, G., Ehn, P., & Kyng, M. (1987) *Computers and Democracy: A Scandinavian Challenge*, Aldershot: Avebury.

(Blomberg et al., 94). Blomberg, J., Suchman, L., Trigg, R. (1994) *Reflections on a Work-Oriented Design Project*, Proceedings of PDC '94, pp 99 – 109, Chapel Hill, North Carolina: ACM Press.

(Brandt, 93). Brandt, S., Madsen, O.L. (1994) *Object-oriented Distributed Programming in BETA,* in Lecture Notes in Computer Science, LNCS 791, Springer-Verlag 1994.

(Brandt, 94). Brandt, S. (1994) *Implementing Shared and Persistent Objects in BETA,* Progress Report, Tecnical Report, Department of Computer Science, Aarhus University.

(Bødker, 91) Bødker, S. (1991) *Through the Interface: a Human Activity Approach to User Interface Design*, Hillsdale, New Jersey: Lawrence Erlbaum Associates.

(COMIC 2.2, 93) COMIC Deliverable 2.2, Esprit Basic Research Project 6225 (1993) *Field Studies and CSCW*, (eds.) Lancaster University and Manchester University.

(Crabtree et al., 97). Crabtree, A., Twidale, M., O'Brien, J., Nichols, D.M. (1997) *Talking in the Library: Implications for the Design of Digital Libraries*, Proceedings of ACM Digital Libraries '97, Phildelphia: ACM Press

(CVS, 98). Gnu, *Concurrent Version System* (1998) ftp://archive.eu.net/gnu/.

(Damm et al., 97) Damm, C.H., Hansen, K.M., Thomsen, M. (1997) *Issues from the GCSS Prototyping Project – Experiences and Thoughts on Practice*, Department of Computer Science, Aarhus University.

(Garfinkel et al., 70) Garfinkel, H. & Sacks, H. (1970) *On Formal Structures of Practical Actions*, Theoretical Sociology: Perspectives and Developments (eds. Mckinney, J.C. & Tiryakian, E.A.), pp 337 – 366, New York: Appleton-Century-Crofts, 1970.

(Greenbaum et al., 91). Greenbaum, J., & Kyng, M. (1991) *Design at Work: Cooperative Design of Computer Systems*, Hillsdale New Jersey: Lawrence Erlbaum Associates.

(Grønbæk, 91). Grønbæk, K. (1991) *Prototyping and Active User Involvement in System Development: Towards a Cooperative Prototyping Approach*. Ph.D. Thesis, Computer Science Dept., University of Aarhus.

(Grønbæk et al., 93). Grønbæk, K., Kyng, M., & Mogensen, P. *CSCW Challenges: Cooperative Design in Engineering Projects*, Communications of the ACM 36 (6), pp 67 - 77.

(Grønbæk et al., In Press). Grønbæk, K., Kyng, M., & Mogensen, P. *Toward a Cooperative Experimental System Development Approach*, In M. Kyng & L. Mathiassen (Eds.), (In Press).

(Grudin, 89) Grudin, J. (1989) *Why Groupware Applications Fail: Problems in Design and Evaluation*, Office: Technology and People, vol. 4 (3), pp 245 – 264.

(Heath et al., 92). Heath, C. & Luff, P. (1992) *Collaboration and Control: Crisis Management and Multimedia Technology in London Underground Line Control Rooms*, JCSCW '92, vol. 1, the Netherlands: Kluwer Academic Publishers.

(Hughes et al., 92). Hughes, J., Randall, D., Shapiro, D. (1992) *Faltering from Ethnography to Design*, Proceedings of CSCW '92, pp 115 – 122, Toronto: ACM Press.

(Hughes et al., 94). Hughes, J., King, V., Rodden, T., Andersen, H. (1994) *Moving Out of the Control Room: Ethnography in System Design*, Proceedings of CSCW '94, pp 429 – 439, Chapel Hill: ACM Press.

(Hughes et al., 96). Hughes, J., Kristoffersen, S., O'Brien, J., Rouncefield, M. (1996) *When Mavis met IRIS: Ending the love affair with Organisational Memory*, Proceedings of IRIS 19 'The Future', Report 8.

(Kensing et al., 97). Kensing, F & Simonsen, J. (1997) *Using Ethnography in Contextual Design*, Communications of the ACM, 40 (7), pp 82 - 88.

(Knudsen et al., 94). Knudsen, J.L., Löfgren, M., Madsen, O.L., Magnusson, B. (1994) *Object-Oriented Environments. The Mjølner Approach*, Prentice Hall.

(Knudsen et al., 96). Knudsen, J.L., Madsen, O.L. (1996) *Using Object-Orientation as a Common Basis for System Development Education*, ECOOP '96 Teachers Symposium.

(Madsen et al., 93). Madsen, O.L., Møller-Pedersen, B., Nygaard, K. (1993) *Object-Oriented Programming in the BETA Programming Language*, ACM Press, Addison Wesley.

(Madsen, 96). O.L Madsen: Open Issues in Object-Oriented Programming (1996) – a Scandinavian perspective, Software Practice and Experience.

(Microsoft, 95). *The Component Object Model Specification*, Microsoft Corporation, 1995.

(Mogensen, 94). Mogensen, P. (1994) *Challenging Practice: an Approach to Cooperative Analysis*, Ph.D thesis, Computer Science Department, University of Aarhus, Daimi PB-465.

(Rational, 98). Rational Software Cooperation (1998) *UML Notation Guide Version 1.1,* http://www.rational.com/uml/html/notation/

(Rouncefield et al., 94). Rouncefield, M, Hughes, J.A., Rodden, T, Viller, S. (1994) *Working with "Constant Interruption": CSCW and the Small Office*, Proceedings of CSCW '94, Chapel Hill: ACM Press.

(Schmidt et al., 93) Schmidt, K. & Carstensen, P. (1993) *Bridging the Gap: Requirements Analysis for System Design*, Working Paper, COMIC-RISØ, Esprit Basic Research Project 6225, (eds.). Lancaster University and Manchester University.

(Twidale et al., 94) Twidale, M., Randall, D., Bentley, R. (1994) *Situated Evaluation for Cooperative Systems*, Proceedings of CSCW '94, pp 441 – 452, Chapel Hill, North Carolina: ACM Press.

(Weiderhold, 97) Weiderhold, J.T. (1997) *A Multi-User Persistence Framework: Building Customised Database Solutions uing the BETA Persistent Store*, MA. Thesis, Department of Computer Science, Aarhus University.

(Wiklund, 94). Wiklund, M. (1994) *Usability in Practice*, AP Professional.

(Wittgenstein, 68) Wittgenstein, L. *Philosophical Investigations*, Oxford: Basil Blackwell, 1968.