THE ISTAR ENVIRONMENT

V. Stenning

Rapporteur: Mr. P. Ezhilchelvan

An Introduction to ISTAR

Vic Stenning

Imperial Software Technology, UK

1. INTRODUCTION

ISTAR is an environment for use on computer systems development projects. It addresses the three critical "dimensions" of such a project - technical development, project management and configuration management - in a coordinated way, and is thereby able to support all members of the project team throughout the project life cycle.

The environment is not specific to any particular development method or programming language, but rather provides a controlled overall structure within which particular methods and languages can be employed as required.

This paper discusses the contractual approach on which ISTAR is based, describes the structure and organisation of the environment, and summarises the available tools and facilities.

2. OBJECTIVES

Concern here is with Integrated Project Support Environments (IPSEs). The scope of an IPSE can conveniently be identified by contrast with a program development system and a programming support environment.

Program development systems have traditionally been supplied by computer manufacturers. They provide only those facilities that are essential to implement programs in some chosen programming language. Thus they would typically offer facilities for editing, compiling, linking and debugging.

Programming support environments recognise that actual implementation in some chosen programming language is only one small part of the complete process of software development. (The term "development" is used consistently here to encompass not only initial development but also subsequent maintenance and evolution during the operational life of the software and system.) They therefore provide facilities to support all development activities throughout the complete life cycle, from initial concept formulation and requirements analysis, right through operational use and into controlled phasing out and replacement. In some cases this system life cycle can last for tens of years. Version control and configuration management are obvious issues that must be addressed by a programming support environment.

Project support environments go beyond programming support environments in that they provide support to all project staff, not just to development staff. Thus they should provide facilities for project management, quality assurance, document preparation, and so on. Ideally, a project support environment should offer facilities for complete system development, not just software. Thus one would expect to see support for total system design methods, with smooth transition into individual design methods for the hardware and software elements.

ISTAR is a full integrated project support environment. In addition to the general requirements for life cycle support and project team support, a number of specific objectives were identified for the product. It should be portable across a wide range of machines, and in particular should be suitable for both shared development machines and single user workstations. Indeed, there must be a smooth transition path from use of shared machines to use of workstations, and this obviously involves a stage of using the two together. It should be open-ended, in that user organisations should be able to incorporate new tools without recourse to the environment supplier. It should support use of existing tools without modification. And it should be able to support projects that are geographically distributed across several sites.

3. THE CONTRACTUAL APPROACH

3.1 The Contract

ISTAR is organised to support a powerful but general approach to software and systems development - the contractual approach. This approach is based upon the hierarchical decomposition of work units into smaller work units that is typically employed for any complex project.

With the contractual approach, each identified task within a project is organised as an individual contract. This contract takes as input a precise specification of the task to be performed, and produces as output the deliverables that are required from the task. Where those responsible for a contract can identify various sub-tasks which would help to achieve the goal, and are able to precisely specify those sub-tasks, then sub-contracts can be let to perform those sub-tasks. The whole structure is of course recursive, and the sub-contracts may themselves have sub-contracts, and so on. The net result is that any task typically involves a complete hierarchy of contracts, where each of those contracts has the same basic form (Fig.1). Within this hierarchy, when a given contract lets a sub-contract, we refer to the former as the "client" and the latter as the "contractor".

As noted above, the main interface between a client and a contractor is that the client supplies a specification to the contractor, and the contractor returns deliverables to the client. However, for coordination purposes the client will typically need to be informed of the contractor's progress and any problems that are encountered. Further, the client may

need to pass to the contractor information that is outside the scope of the specification, for example some informal response to a problem report. Thus there may be a flow of reports in both directions between client and contractor, and the complete client-contractor interface has three components: specification, deliverables, and reports (Fig.2).

3.2 The Contract Specification

A contract specification is regarded as having three distinct parts

- a task specification, which precisely defines the task to be performed (what rather than how)
- a set of verification criteria, which define an objective test to show that the task has been performed satisfactorily
- and a set of management constraints that govern the performance of the task. These may cover, for example, the required timescale for the task, resources to be employed, standards to be applied, and so on.

Of course, the nature of the contract specification will vary with the task to be performed, and different kinds of specification will be appropriate at different levels of the hierarchy. Thus, for a top level step that encompasses a major product development the task specification would typically concentrate on the market requirement and the verification criteria might call for acceptance testing according to established procedures. However, for a low-level step the task specification might provide a detailed interface specification for a software module to be implemented in Pascal, and the verification criteria might define a specific set of tests to be performed on the implemented module.

3.3 Amendment and Cancellation

It cannot be assumed that all contracts will proceed smoothly and produce their deliverables as specified and within the management constraints. Some contracts will take longer than planned, or consume more than the allocated resources. It might prove impractical or impossible to produce deliverables that meet the specification. Or the need to revise the contract specification may arise externally.

Therefore it must be possible to amend contract specifications. However, such amendments can only be made by the client. Should a problem arise within a contract then this must be reported to the client (using the normal reporting facilities) who may choose as a result to amend the contract's specification. The contractor may use reports to suggest or negotiate contract amendments, but cannot unilaterally make such amendments. By contrast, the client can make an amendment at any time - and of course must take the responsibility for doing so.

Occasionally, due to changing circumstances or

insurmountable problems, it may become pointless to continue with a contract. In this case the client may choose to completely cancel the contract.

3.4 Specification Issues

All main aspects of the contractual approach have now been introduced - the letting of contracts, exchange of reports, return of deliverables, amendment of contracts, and cancellation. However, two further points should be made concerning contract specifications.

First, although the specification must be precise, it need not be detailed. Thus, for example, a contract specification may call for a feasibility study to be performed, but may not detail the options to be investigated. The "rules of the game" are that any deliverables that meet the specification and satisfy the acceptance criteria are legitimate. Thus it is the client's responsibility to provide an appropriate specification of sufficient detail to ensure that the returned deliverables will be satisfactory. Of course, the client could misjudge the level of detail that is needed, and as a consequence receive an unsuitable deliverable. In this event it is necessary to produce a more detailed specification, removing the area of freedom that allowed the unsatisfactory deliverable, and then issue the appropriate contract amendment (or perhaps even let a completely new contract).

The second point on the contract specification concerns the acceptance criteria. It would have been possible to regard the definition of acceptance criteria as part of the task specification. However, by choice the acceptance criteria are separated out, both to emphasise their importance and to indicate that an objective means of assessment should be defined before a contract is let, rather than while it is proceeding. Of course, it may later prove necessary to modify the acceptance criteria, but this must then be treated as a contract amendment.

4. USING THE CONTRACTUAL APPROACH

The contractual approach reflects a common way of organising projects that is completely general. It corresponds to the "work breakdown" approach that is typically employed (consciously or unconsciously) for any non-trivial project. The objective in following this approach is primarily to instil a basic level of project hygiene and to ensure that, at all times, all the people involved in a project know exactly what they are trying to do.

To appreciate the generality of this approach, first consider an organisation that typically conducts its projects in phases: feasibility study, requirements analysis, system specification, and so on. Within ISTAR the complete project would be a contract, and this contract would then let one sub-contract for each phase. These sub-contracts would themselves let sub-contracts as appropriate.

Of course, it is frequently the case that the different

phases are not strictly sequential. Rather, the work on a given phase can often be initiated as soon as the previous phase has produced useful output. This again can be accommodated within the contractual approach. Each sub-contract is now required to produce not just a single deliverable, but rather a set of deliverables. The sub-contract for a new phase is initiated as soon as the previous phase produces a useful deliverable. This sub-contract must then be amended as further relevant information becomes available, but with proper planning these amendments can be handled without disruption. Only in the case where there is a genuine change of requirement or design need there be any significant re-working, and in these cases such re-working is inevitable. Obviously this parallel working with overlapping phases requires more coordination than the sequential case, and as always this coordination must be the responsibility of the client contract.

Within a given phase it is often possible for work to proceed in parallel. The classic example is where a system can be decomposed into component parts and, once specified, each of these components can be developed independently. With the contractual approach, the decomposition into components, and the specification of these components, is performed within a coordinating contract (or by a sub-contract on behalf of that contract). The coordinating contract then lets sub-contracts for the production of the individual components, with all these sub-contracts proceeding in parallel. Any interfacing problems that subsequently arise must be handled by the coordinating contract, and this may of course involve amendments to various sub-contracts. Eventually the required deliverables will be returned by all the sub-contracts, and these can be combined to yield the desired system.

Discussion thus far has been on the basis of sequential phases, possibly with parallel development within the phases. However, the contractual approach is obviously not limited to such an arrangement, and in general any required combination of sequential and parallel working can be employed. This is achieved by letting sub-contracts at the appropriate times and with the appropriate management constraints, particularly on timescales. The degree of parallel working is constrained only by practical considerations of retaining overall control and avoiding excessive amounts of rework.

Consider now some extensions to the basic scenario. Suppose that, in order to assist with requirements analysis or design, it is decided to construct a rapid prototype. This is obviously handled by letting a sub-contract, with the deliverable either being the prototype itself or the results of building and experimenting with the prototype, whichever is most appropriate.

Now suppose there is a need to construct a product and, because of time constraints, to simultaneously develop a user guide for that product. This might best be handled by separate contracts, one for product development and one for the user guide, with deliverables from the former being fed to the latter as they become available. In this case it might be appropriate for the specification of the user guide contract

to be expressed in terms of "reflecting the current state of knowledge of the product", so that the contract would not need amending every time more information became available.

Finally, suppose that it is desired to develop a new product and simultaneously develop a set of acceptance tests for that product, both being based upon the same initial specification. Again separate contracts will be let, one for the product and one for the acceptance tests, but there are in fact several possible ways of proceeding. However they all involve initially defining some "working" acceptance criteria for the production contract. One possibility would be to allow the production contract to proceed to completion on the basis of these working criteria, and then let a separate contract to run the independent acceptance tests. Should any of these tests fail the production contract could be amended to reflect the detected problems, and the tests re-run on the subsequent deliverable. Other approaches are also possible, and these could be equally viable.

Obviously the above discussion has not been exhaustive. The intention was simply to illustrate the generality of the contractual approach and its relationship to some recognised project organisations. As stated earlier, the objective of this approach is primarily to encourage basic project hygiene and to ensure that the people working on a project know precisely what they are trying to do.

5. THE ORGANISATION OF ISTAR

ISTAR is based upon the contractual approach, and its organisation directly reflects that approach. This perhaps has its greatest impact in the area of the database. Rather than having one large "environment database", ISTAR employs a large number of small databases, one for each contract. As a new contract is let, the database to hold information pertaining to that contract is created automatically. The relationships between the individual databases, reflecting the contract hierarchy, are maintained by ISTAR.

It is on the basis of these small databases that an ISTAR system can be geographically distributed. Individual databases within the same contract hierarchy can be held on different machines. A single contract database must be held in its entirety on a single machine, but related databases - for example, the databases for two contracts where one is a sub-contract of the other - can reside on different machines. Thus the complete contract hierarchy within a given ISTAR system can be dispersed over an entire network.

All the basic operations of the contractual approach, as summarised at the beginning of section 3.4, are directly supported as ISTAR primitive operations. All these contractual operations (except reporting) involve a "handshake" exchange between client and contractor. Thus a new contract is let by a client assigning the contract to some user of the environment, and that user must subsequently acknowledge the assignment. That particular user thereby accepts overall responsibility for the contract, although other users may work on the contract as required.

The contractor can subsequently make deliveries, and these are acknowledged by the client. Similarly the client may either amend or cancel the contract, and again these operations require acknowledgment from the contractor. And, of course, reports can be sent in either direction at any time for any extant contract.

6. TRANSFER ITEMS AND CONFIGURATION ITEMS

Internally, an individual contract database is partitioned into a number of distinct areas. Specifically, each contract database has precisely one "contractual" area and an arbitrary number of "work" areas (Fig.3). As these names suggest, the contractual area is used primarily for coordination with other contracts, while the work areas are used for performing work within this contract.

Two types of information unit are particularly important within ISTAR, namely the transfer item and the configuration item. A transfer item is a single self-contained unit of information of a given type: META-IV specification, Pascal source, project plan, document, or whatever (Fig.4). A configuration item is a set of transfer items (Fig.5).

Information is held in the contractual area in the form of configuration items, and it is configuration items that are moved between contract databases. (Such moves are achieved by copying, so that following such a move the configuration item exists both in the source database and in the destination database.) Thus, when the contract is established, its specification is installed as a single configuration item in the contractual area. Similarly, any subsequent amendments are also installed as single configuration items within this area, with a relationship to the original specification and earlier amendments. And a deliverable from the contract must be established as a configuration item in this area before the actual delivery to the client can be made.

Similar arrangements apply for any sub-contracts that may be let. Thus, the specification of a sub-contract will be established as a single configuration item in the contractual area before the sub-contract is assigned. Deliverables from the sub-contract will be installed in this area as they arrive. And so on.

Once established in the contractual area, configuration items and their member transfer items are normally immutable. Work within the contract does not modify established configuration items, but rather produces new configuration items - the contract deliverable, for example, or a new sub-contract specification. This is done by first creating an empty configuration item in the contractual area, and then developing various transfer items in various work areas and "exporting" these transfer items to the contractual area. In order to produce a new transfer item it may be necessary to consult or employ some existing transfer item - from the contract specification, for example - and these can be imported into work areas as required (Fig.6).

Thus configuration items are held in contractual areas and moved between contract databases, while their member

transfer items are imported from the contractual area into work areas, or exported from work areas into the contractual area.

Within the contractual area, both configuration items and transfer items within configuration items can exist in many distinct versions. A simple naming scheme is adopted, whereby there are distict variant "threads" for each item, with many successors within each thread. A particular version of an item is then identified by specifying the variant and the successor, thus: STACK_SPEC(UNBOUNDED,5). Various naming defaults can then be employed when accessing existing versions, for example to access the latest version or some preferred version. Although the naming scheme is deliberately kept very simple, the data management facilities recognise a richer versioning structure, involving arbitrary trees, and record this structure by means of relationships within the contractual area. These relationships can then be queried and used where appropriate by users or tools.

The discussion thus far has perhaps suggested that configuration items can only move up or down the contractual hierarchy, between client and contractor. However this is not in fact the case. Rather, a configuration item can be moved on request from any database to any other database, subject only to access right restrictions. Such moves are normally recorded at both databases, with the source recording the destination and the destination recording the source. Thus detailed records of the movement of configuration items are maintained.

This general facility for moving configuration items between databases is employed extensively within ISTAR. For example, when there is a need for a component library this is achieved by establishing a contract to operate the library. Library components are of course configuration items. New components may be submitted to the library from any source, and the source of each component is recorded. Contracts may take components out of the library as required, and all usage of a given component is again recorded. Defect reports can easily be sent to the original donor and all users of a given component, and any new version can readily be distributed to all interested users.

7. WORKBENCHES

The many tools within ISTAR are not simply organised as one large toolkit. Rather, the tools are grouped into collections of related tools, termed "workbenches". Each workbench typically operates on just a few transfer item types that are in some way related. For example, a simple workbench that supports development in some programming language might operate on two transfer item types: source module in that programming language, and executable program.

As might be expected, a workbench typically operates in its own work areas within the contract. Thus a Pascal workbench would operate in Pascal work areas, a VDM workbench in VDM work areas, and so on. The workbench would support import and export of transfer items of the relevant types, and

analysis and production of items of these types. Each kind of work area - Pascal, VDM, or whatever - has a well-defined "data model" that governs the organisation of data within such work areas. This data model is defined solely to meet the needs of the workbench, and is independent of the data model for the contractual area or that of any other work area.

Indeed, achieving this independence of data models was a major objective of the contractual area and work area arrangement. A work area is completely self-contained and insulated from the outside world, to which it interfaces solely by (workbench specific) import and export operations. This means that workbenches can be developed independently and incorporated into ISTAR without danger of clashing with existing workbenches. This is obviously important to user organisations that wish to extend the system, and is particularly important when incorporating existing tools that impose their own requirements on the organisation of data. In the latter case, a new kind of work area is introduced, with a data model conforming to the requirements of the existing tool. The tool is then incorporated into a workbench that operates on this kind of work area.

Transfer items are typed, in the sense that they will be processed only by workbenches designed to operate on items of that type. In this context, it should be noted that the contractual area and the contractual operations are completely independent of transfer item types. A useful analogy is that of shipment of standard containers on lorries or ships. A workbench can process the contents of a transfer item, but as part of the export operation this transfer item is loaded into a standard container that is then labelled with the type of the transfer item. These containers can be held in contractual areas, and moved between databases, without any need to examine their contents. However, when these contents are required in some other work area the container is unloaded into that work area as part of the import operation. Of course, such unloading will only be performed by a workbench capable of processing transfer items of that particular type.

8. THE USER INTERFACE

All ISTAR workbenches and tools interact with the user via a common user interface system. This user interface provides a range of facilities, to be used by workbenches as appropriate

- full screen editing
- multiple windows
- pop-up menus and windows
- forms with protected fields
- syntax-directed editing

In addition, an extended version of the user interface system supports graphical display (see section 10.5).

The user normally directs the system by means of menu selection. Direct entry of commands is also possible.

Since there is a single common user interface package, all editing commands are common throughout the system and for the different modes of editing. Thus, screen editing, forms editing and syntax-directed editing all employ the same basic set of commands.

9. THE USER'S VIEW OF THE SYSTEM

As might be expected, the user's overall view of ISTAR is dominated by the contractual structure.

When a user first logs in to the system, the log-in display presents basic information on all contracts in which that user has some involvement. Specifically, the display lists the established contracts for which the user has some responsibility, highlights for each such contract any significant events that are awaiting acknowledgment, and also indicates any new contract assignments to this user. Recall that a user can have some responsibility on a given contract either because that contract was initially assigned to that user, or because that user was subsequently given a task to be performed within the contract. A significant event for a contract is the arrival of an amendment, a deliverable from a sub-contract, any kind of report, an incoming configuration item from another database, or a cancellation order.

As an example, the large window in the log-in display shown in Fig.7 indicates that the user is currently responsible for three contracts, called "ddtest", "dd_ug", and "feas_rpt". This latter contract has been cancelled by its client, and this user has not yet acknowledged the cancellation.

Typically, having examined the log-in display the user will select a contract on which to work. Selection of "ddtest" from Fig.7 leads to the display of Fig.8, where the window dedicated to "ddtest" indicates that this contract has been opened.

Because ISTAR offers a large number of workbenches they have been grouped into five categories. Selection of the "function" option within the "ddtest" window pops up a menu listing these categories. Selection of a category, such as "technical development", then pops up a menu listing the workbenches in this category. An individual workbench can then be selected from this menu to operate on a work area within contract "ddtest". A workbench would typically employ the whole screen for its interactions, with similar usage of windows and pop-up menus, and on exit from the workbench the display would revert to that shown in Fig.8.

10. AVAILABLE WORKBENCHES

An ISTAR workbench typically operates within a single work area and interacts with the remainder of the system by importing and exporting transfer items. Workbenches normally employ a number of discrete tools, but the boundaries between the individual tools are often obscured from the user. The objective of a workbench is not to present a set of disjoint tools, but rather to provide a coordinated range of facilities for performing work of a given kind. Thus a complete workbench, including its component tools, is designed and presented to users as a single coherent whole.

As mentioned in the previous section, ISTAR workbenches are grouped into five categories: general, project management, technical development, configuration management, and tool development. These categories are discussed individually below.

10.1 General

The "general" category includes three workbenches: text, documentation, and timesheet completion. (A personal mail facility is also provided by ISTAR, but this facility is generally available throughout the system, rather than provided by a specific workbench.)

The text workbench offers simple word processing facilities, and is used for preparing transfer items of type "text". This is a common transfer item type, since most specifications and deliverables feature a text item that summarises their more formal content.

The documentation workbench offers similar word processing facilities, but recognises the concept of a document that has chapters and sections. These partial documents can be held in several versions, and a complete document can be assembled from selected versions. A spelling checker is also provided.

Inclusion of text processing and mail facilities reflects the philosophy that all personnel on a project should regard ISTAR as their normal working environment. Much of the work on any project is concerned with preparation of documents, and personal mail is now in common use. ISTAR therefore provides facilities in these areas so that users can remain within the environment, with no need to invoke some other system or tool.

The timesheet completion workbench supports the filling and submission of weekly timesheets. This workbench is closely related to the resource management system, which is discussed below.

10.2 Project Management

The project management category includes two workbenches: contract management and resource management.

The contract management workbench directly supports the management of an individual contract and its sub-contracts. Specifically the workbench includes tools for work breakdown structuring, estimation, scheduling, detailed task definition,

and progress monitoring. These tools are used in combination to support the activities of coordination, planning and monitoring.

The work breakdown tools supports the decomposition of a given task into its component sub-tasks and the identification of dependencies between those sub-tasks. The estimation tool provides estimates of the effort profile for a task, using the COCOMO model. The scheduling tool produces plans for completion of a task, based upon its work breakdown structure and a knowledge of available resources. The task definition tool allows a task within a work breakdown structure to be specified in detail, covering task specification, verification criteria and management constraints (section 3.2). And the monitoring tool produces performance reports for a task based upon reported progress and actual resource usage of its component sub-tasks.

The workbench can produce both textual and graphical reports, with the latter being used, for example, for the presentation of PERT networks.

The contract management workbench is in many ways comparable to a "conventional" management toolkit, albeit a particularly good one. It would therefore have been possible to build this workbench from some existing set of tools, exploiting ISTAR's ability to run such tools without modification. However, it was decided to build new tools specifically for ISTAR, because we wanted the management workbench to be very closely coordinated with the overall contractual structure. Thus, for example, the scheduling tool interacts closely with resource management centres (see below), new sub-contracts can be let directly from the task definition tool, the monitoring tool monitors progress on sub-contracts as well as on local tasks within this contract, and so on.

The resource management workbench is used to control the deployment of people and other resources across contracts, where these contracts may be in different project hierarchies. Any individual resource that is to be managed within ISTAR is affiliated with a specific resource management centre. These centres are created in the normal way, by letting contracts, and a given ISTAR system can have as many centres as required. Thus, for example, when a given organisation is sub-divided into groups - a communications group, a user interface group, and so on - there could be one centre corresponding to each group.

Each centre maintains records of the types and current allocations of the resources under its control. When a contract requires a resource of a particular type it obtains this resource from an appropriate centre, which then records the new allocation.

The resource management workbench interacts closely with the project management workbench and the timesheet completion workbench (section 10.1). The scheduling tool requests information on resource availability from resource management centres, and subsequently forwards bookings for resources whose use has been scheduled. The task definition tool notifies appropriate resource management centres when each

task is activated, and of any subsequent change to the status of the task (e.g. when it is completed). The timesheet collection tool forwards completed timesheets to the appropriate resource management centre for validation against active tasks, and the centre then forwards validated entries to the monitoring function of the appropriate contracts. And if a contract shows signs of falling behind schedule the monitoring tool may request data on resource allocations in order to perform a forward projection.

10.3 Technical Development

The technical development category includes workbenches for CORE, SDL, VDM, Pascal and Unix/C(*).

CORE is a method of requirements analysis that places considerable emphasis on a "whole world" view (rather than modelling just a system's interfaces or internal operation) and on extensive analysis of the emerging "world model". It is a genuine method with a well-defined procedure to be followed. The ISTAR workbench supports all steps of the method and provides a very extensive range of analyses, going beyond those that are conventionally associated with manual use of the method to incorporate some powerful checks that are only feasible with an automated tool operating on a model residing in a database.

SDL (System Description Language) is the CCITT-recommended specification notation for concurrent systems. A system is modelled as a set of "blocks" that communicate with each other and with their environment by exchanging "signals" over "channels". Blocks can recursively be decomposed into sub-blocks. At the lowest level the blocks contain processes that receive and send the signals of the block. An SDL process has a discrete set of "waiting states" where it is awaiting an incoming signal. When such a signal arrives the process performs a transition to a new waiting state; during this transition the process would normally perform some computation and perhaps send some signals. The ISTAR workbench supports progressive decomposition with consistency checking, definition of processes at the lowest level, and code generation directly from the process definitions. The latter two facilities are actually provided by the SX1 tool, developed by British Telecommunications, which has been integrated into the ISTAR workbench.

VDM (Vienna Development Method) is a formal development method for sequential programs, with strong emphasis on abstract data types. The method supports both initial specification and sequential refinement from this initial specification, if required with formal verification at every step. The ISTAR workbench currently provides only limited support for the method; specifically it supports syntax-directed editing of the method's specification language (META-IV) and simple type and signature checking of this language.

As discussed in section 7, the design of ISTAR allows existing tools to be incorporated into workbenches without modification. This was exploited in the case of the SDL

workbench, where SX1 was incorporated, and is also exploited in the case of workbenches that support implementation languages. Thus, both the Pascal and Unix/C workbenches are based upon pre-existing compilers and other tools. The Pascal workbench supports syntax-directed editing and compilation. The Unix/C workbench is simply one that provides direct access to the facilities of Unix; because of the close association in this case between language and operating system there is no separate "C workbench". An Ada workbench is currently under development.

The technical development workbenches that are currently available reflect the initial interest in one particular application area, namely that of real-time systems. However, the overall design of ISTAR is in no way specific to that application area, and other areas could be supported by providing appropriate workbenches in the technical development category. For example, consideration is being given to a workbench for SSADM, a method that is typically employed for the design of DP systems.

10.4 Configuration Management

There are two workbenches in the configuration management category, namely component management and build.

It should be emphasised that the basic configuration management facilities of ISTAR - version identification, freezing of items, tracking of item usage and movement - are not the responsibility of any individual workbench. Rather, these facilities are "built in" to the underlying structure, and are pervasive throughout the system.

Thus the configuration management workbench does not implement the basic versioning and control mechanisms, but rather is more concerned with administrative issues. Specifically the workbench supports such operations as setting and querying preferred versions, querying version histories, establishing and querying relationships between items, and so on. It should be noted that all such operations are available to other workbenches, and indeed these operations would normally be performed by tools as part of their normal function rather than explicitly by the user. However, the configuration management workbench provides a direct user interface to these operations, should this in some circumstances be required.

The configuration management workbench also provides support for component libraries, as discussed at the end of section 6, and for the submission and control of problem reports. The facilities in these areas are heavily dependent upon the more general configuration management facilities, and these functions are therefore included in this workbench for reasons of user convenience.

The build workbench supports the construction of new transfer items by applying tools to existing transfer items. An obvious special case is the production of some required "system" by integration of its component sub-systems. However, the workbench is not limited to this special case. In ISTAR it is common for a contract deliverable to be formed

by combining deliverables from sub-contracts, and this applies whether the required deliverable is a program, a specification, a document, or whatever. In all these cases the build workbench would be used to construct the deliverable.

Basically, the workbench is given a "construction plan" for the required construction process and a "bill of parts" identifying the specific transfer items to be input to that process. The workbench then constructs the required transfer item(s) and also generates a precise record of the build. This record serves both to show the dependencies between transfer items and also as a possible input to subsequent builds. For example, suppose that a new version of one of the input transfer items is produced and it is required to re-run the build using this new version as an input but with all other inputs remaining unchanged. This can be done simply and reliably by using the record from the previous build.

10.5 Tool Development

The tool development category includes three workbenches: APCR, interface definition, and ARLO. As the name suggests, the workbenches in this category support the development of new ISTAR tools and workbenches. These workbenches are therefore of interest to those who wish to extend the system to support a particular method or address a particular need.

The APCR (analyser/prompter/checker/reporter) workbench is used to develop new workbenches to support specific "structured" methods. There are a large number of such methods - SADT, SA/SD, and so on - each with their own particular features but all with a great deal in common. Essentially, any structured method involves construction of a model of the desired system using a small number of entity types and relationship types. Typically such models are presented graphically, with entities of different types being represented by boxes of different shapes and relationships being represented by lines between boxes. A particular method defines the entity and relationship types to be employed and a sequence of stages for constructing the model, typically with specific checks to be performed at each stage.

In ISTAR, all structured methods are supported in the same way. The model is held explicitly in a database, with entities and relationships in the database corresponding to those in the model. Checks on the model are implemented by running analysis programs on the database, and reports are generated from the database.

The APCR workbench generates other workbenches to support particular structured methods. The user of the APCR workbench is prompted for a definition of the method to be supported, in terms of its entity and relationship types, the stages of the method, and the prompts and checks associated with each stage. The APCR workbench then generates a new workbench that supports the various stages of the defined method.

An example of the use of the APCR workbench is provided by the CORE workbench, which was generated in this manner. Further, the method used for defining structured methods is

itself a structured method, and the facilities of the APCR workbench were therefore used to generate the workbench - in much the same way that compilers are bootstrapped and eventually used to compile themselves.

With most structured methods it is desirable to present various "views" of the model in graphical form. This can be done by using the graphics facilities of the extended user interface system. However, this requires the production of a "filter" that extracts the appropriate information from the database and presents it to the user interface system in a generic form. At present such filters are implemented by writing a program, making extensive use of a database query language, or by using ISTAR's report generator. Typically, new filters are produced by modifying some existing filter, rather than by starting "from scratch". It would be possible to largely automate filter production, and a workbench to do so may be produced in the near future.

The graphics presentation facilities are not restricted to use in conjunction with the APCR kit, but are generally available. For example, the contract management workbench uses the graphics interface for presenting PERT networks, and the SDL workbench uses the interface for presenting block hierarchies.

The interface definition workbench relates to the forms and syntax-directed editing capabilities of the user interface system. For each kind of form to be edited the user interface system requires a table defining the form. Similarly, syntax-directed editing requires tables defining the syntax of the language and required layout. When introducing a new form or language the interface definition workbench is used to generate the required tables from, respectively, a forms definition notation or an augmented BNF notation.

The ARLO workbench can be used to rapidly develop new workbenches and individual tools. ARLO is an interpretive language specifically designed for easy development of interactive tools. Using the language it is possible to quickly produce a working prototype or production tool and then incrementally improve and extend the tool as desired. ARLO is also useful for incorporating existing tools into an ISTAR workbench. A major problem with such tools is that they do not operate on ISTAR databases, but rather on files. This problem, and the problem of user interface consistency, can be addressed by wrapping the existing tool in an "envelope". This envelope initially interacts with the user and the appropriate ISTAR database, sets up access to the required files, and then invokes the existing tool. Upon return the envelope updates the ISTAR database as appropriate, dependent upon the completion status of the tool. ARLO is a convenient language in which to implement such envelopes.

The workbenches in the tool development category are delivered to users as an integral part of the system. This is in keeping with the overall objective that user organisations should themselves be able to extend the system to meet their own particular needs.

11. STATUS

The ISTAR system, including the workbenches discussed in section 10, is available from Imperial Software Technology as a commercial product.

The system is currently implemented under Unix. It will run under any "real" Unix (as opposed to "Unix-like"), including System V and BSD 4.2. It has been ported to several machines, including VAX, Pyramid, AT&T 3B2, and 68000-based workstations. Ports to other machines with a real Unix are straightforward. A port to VAX/VMS(**) is scheduled during 1986.

Communications facilities are not implemented as part of ISTAR itself. Rather, the system is interfaced to whatever communications facilities are available. The only requirement is that the communication medium should be able to transfer a file (i.e. a large block of data) from one machine to another with a reasonable level of reliability. The current implementation communicates using any combination of Ethernet TCP/IP, RS232 using UUCP, and physical transfer of magnetic media. ISO protocols will be supported as soon as a suitable Unix implementation becomes available.

For its graphics, ISTAR uses GKS.

12. FINAL REMARKS

ISTAR is a rich environment, and inevitably the latter parts of this introduction have concentrated on the available workbenches and tools. However, the path to success with ISTAR does not lie with making optimum use of some individual facility. Rather, it is important to make effective use of the system as a whole, and particularly to exploit the overall contractual structure.

Thus any consideration of ISTAR should not begin at the level of individual tools or facilities, or with details of the database system or the user interface. It should instead begin with consideration of the contractual structure and the way in which this can be deployed to achieve overall project control and visibility and to ensure a basic level of hygiene. This in itself can make a major contribution in the areas of quality and productivity, and is also an essential prerequisite to the introduction of better methods and tools that can offer further improvements.

ISTAR addresses the concerns at both levels - overall structure and hygiene, and individual methods and tools - but it is important that these concerns are taken in the right order: first overall structure, then methods, then tools. This was the order in which ISTAR was designed, and it is the order that should be followed in any consideration or use of the system.

13. TRADEMARKS

(*) Unix is a trademark of AT&T Bell Laboratories
(**) VAX and VMS are trademarks of Digital Equipment Corporation

14. ACKNOWLEDGMENTS

ISTAR is a collaborative development by Imperial Software Technology and British Telecommunications PLC, and many people at both organisations have contributed to its design and implementation.

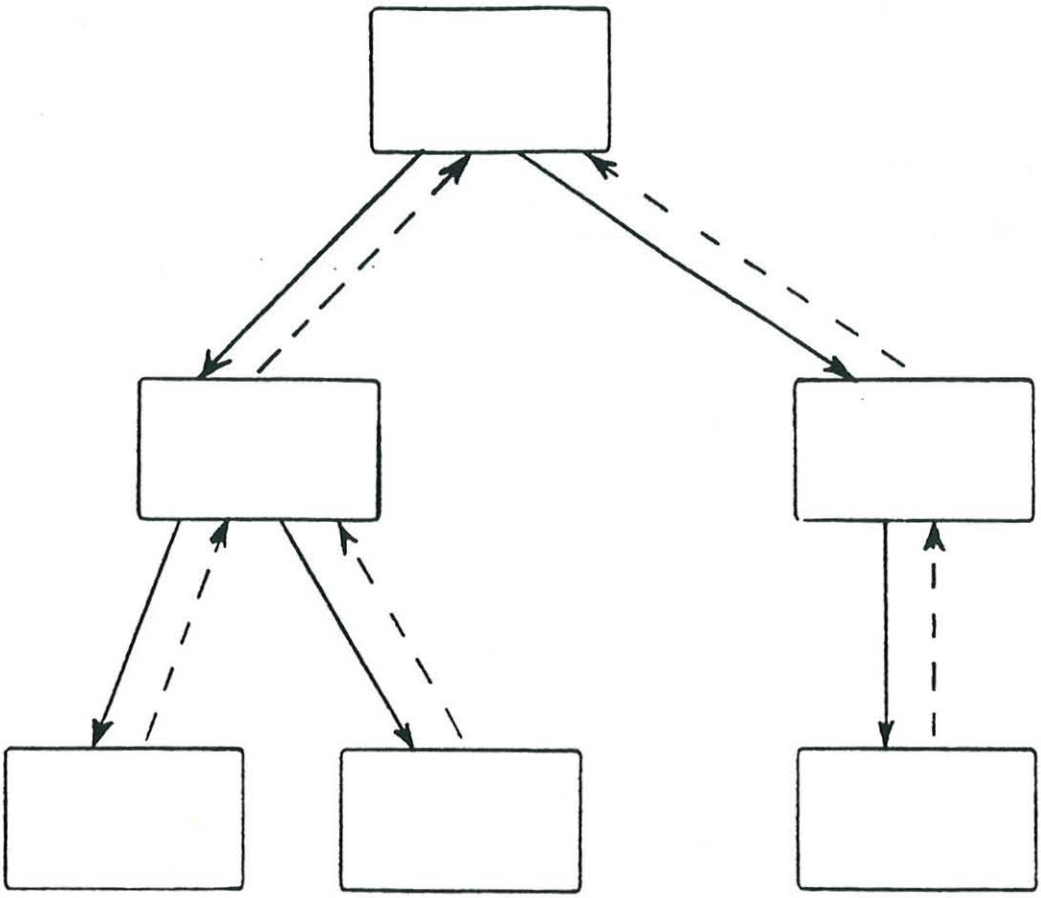

Fig. 1 A contract hierarchy

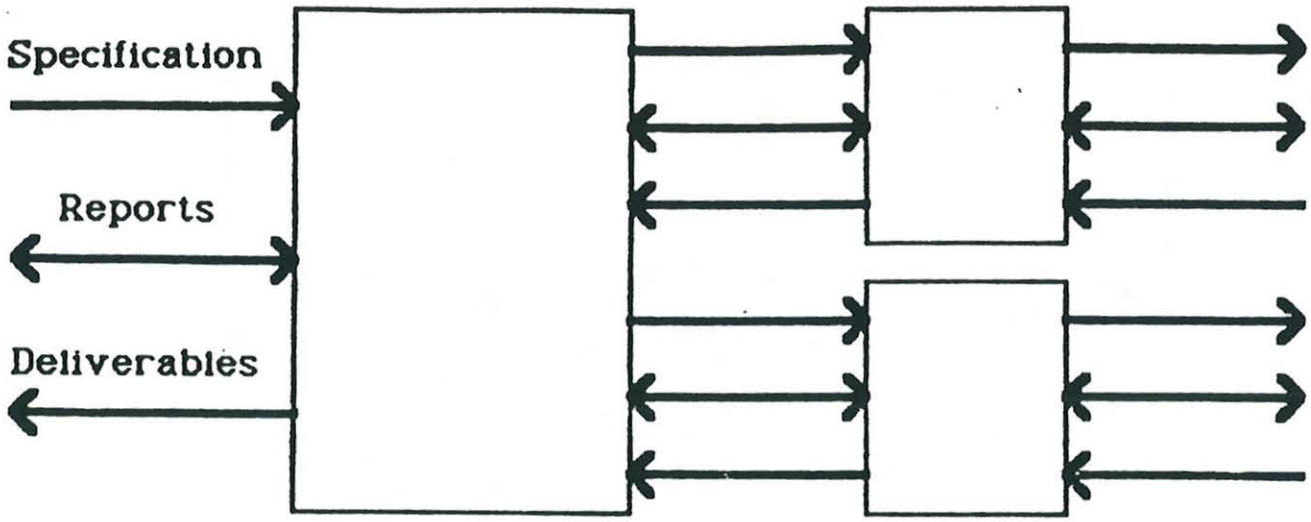

Fig 2 The client/contractor interface

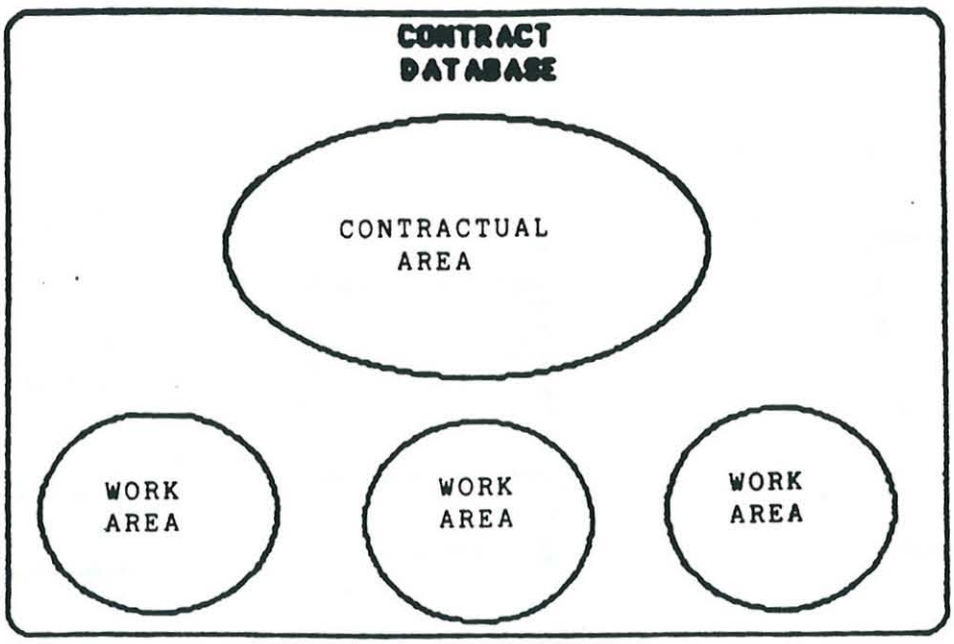

Fig. 3 A contract database

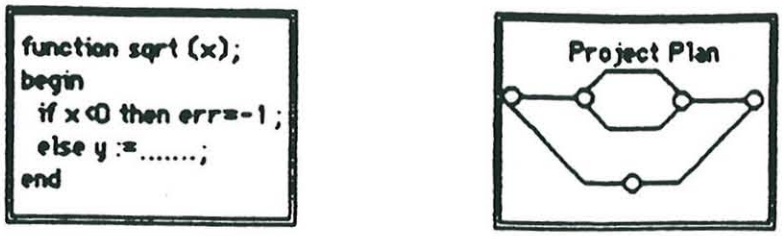

Fig. 4 Transfer items

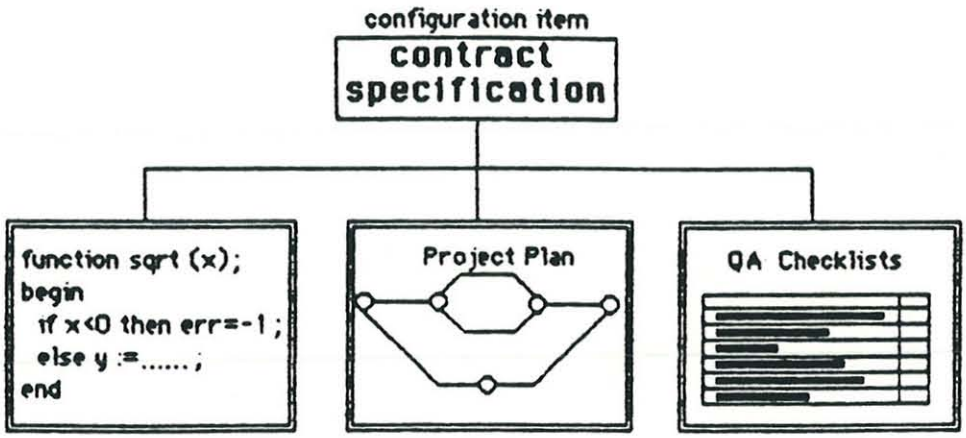

Fig. 5 A configuration item

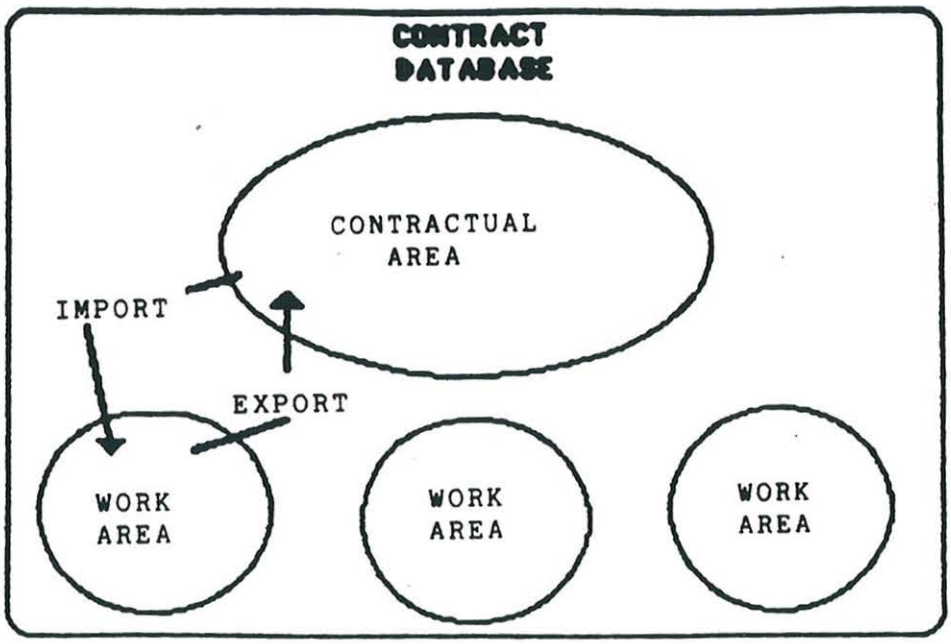

Fig. 6 Import and export

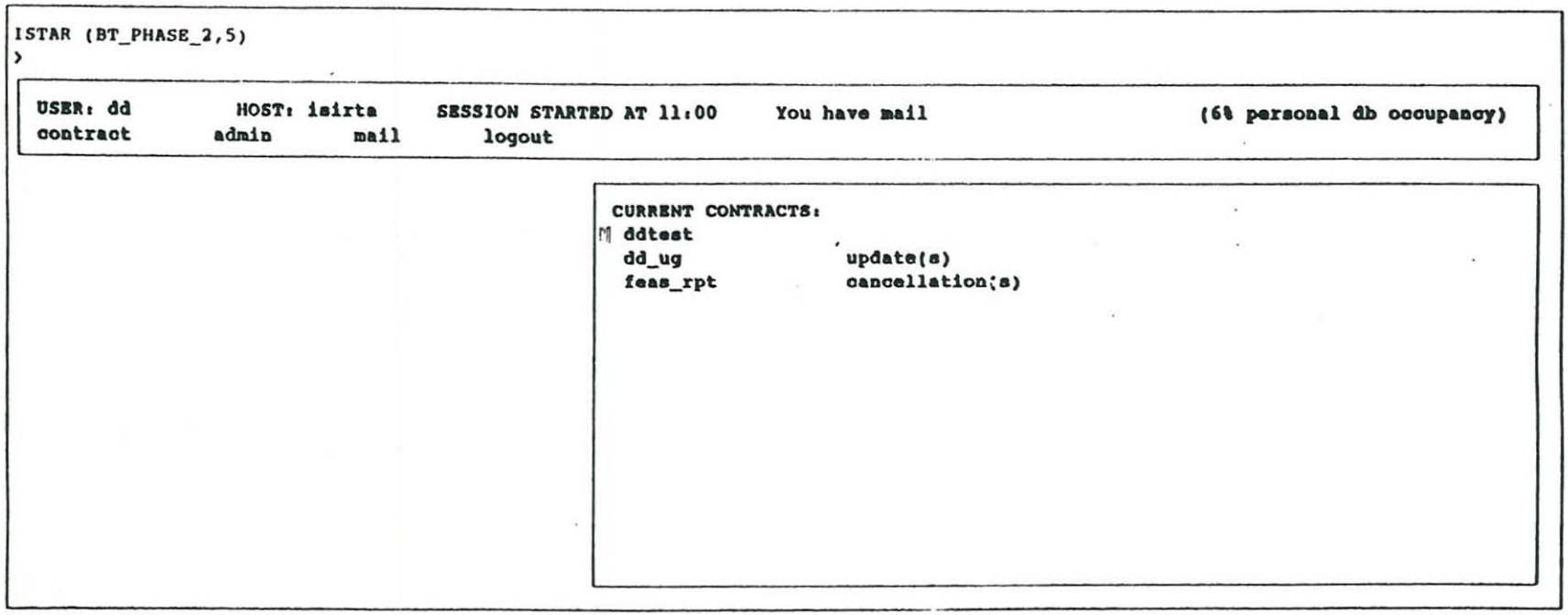

Fig. 7 A log-in display

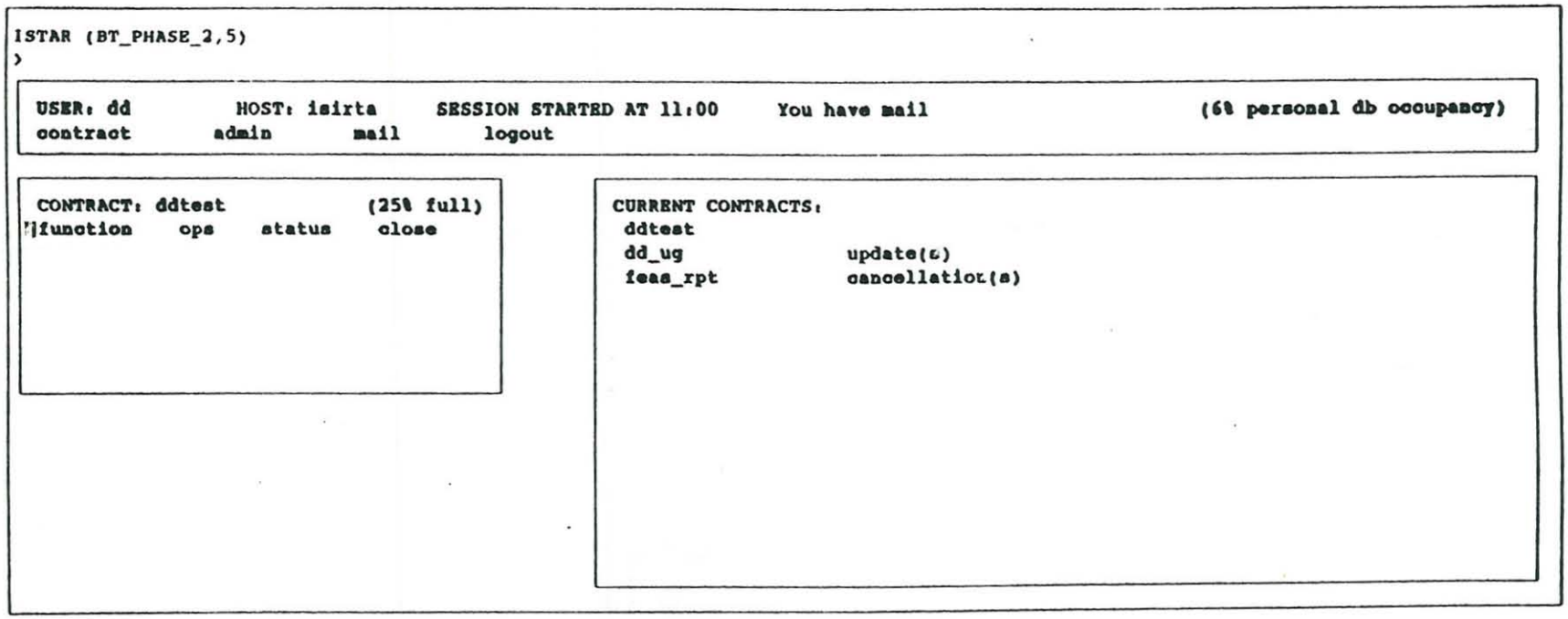

Fig.8 A selected contract

DISCUSSION

The post talk discussions were sparked off by Professor Brown's question on development costs and the market prospects of the ISTAR project support environment. The speaker replied by admitting that the development was expensive; since the project itself is in the early stages of being under experimentation and evaluation, he found it hard to predict its market potential. However, he quoted a few organisations that are interested in buying the tools.

Professor Kopetz was interested in knowing if any implementations has been done using the contractual approach. The speaker mentioned IST and other organisations using his methodology.

Professor Habermann was concerned about training programmes for personnels working with the ISTAR. The speaker replied that such a training programme was being run in collaboration with British Telecom.

Professor Atkinson asked about the threshold size of the projects the speaker would recommend for the ISTAR environment. The speaker replied that the threshold size tended to be small.