THE ASPECT INFORMATION BASE

P. Hitchcock

Rapporteur: Mr. R. Weedon

Introduction

Aspect is a project funded by the Alvey Directorate. The collaborators are System Designers Limited, MARI, ICL and the Universities of York and Newcastle upon Tyne. The project started officially in April 1984, with initial funding for three years.

The objectives of Aspect are to provide an integrated project support environment which will run on distributed host machines that are geographically separate. It will support programming in Ada, Pascal and C and give the ability to develop code for distributed target machines. Two interfaces are defined: a public tool interface which allows new tools to be written and included in the system, and an open tool interface. The open tool interface allows the use, as far as possible, of the existing body of Unix tools. This need to support a wide variety of tools, and the provision of the Public Tool Interface, led to the concept of Aspect being an ipse-kit. That is to say, it is a general purpose framework that can be populated with a particular tool set and become a customised ipse.

Architecture

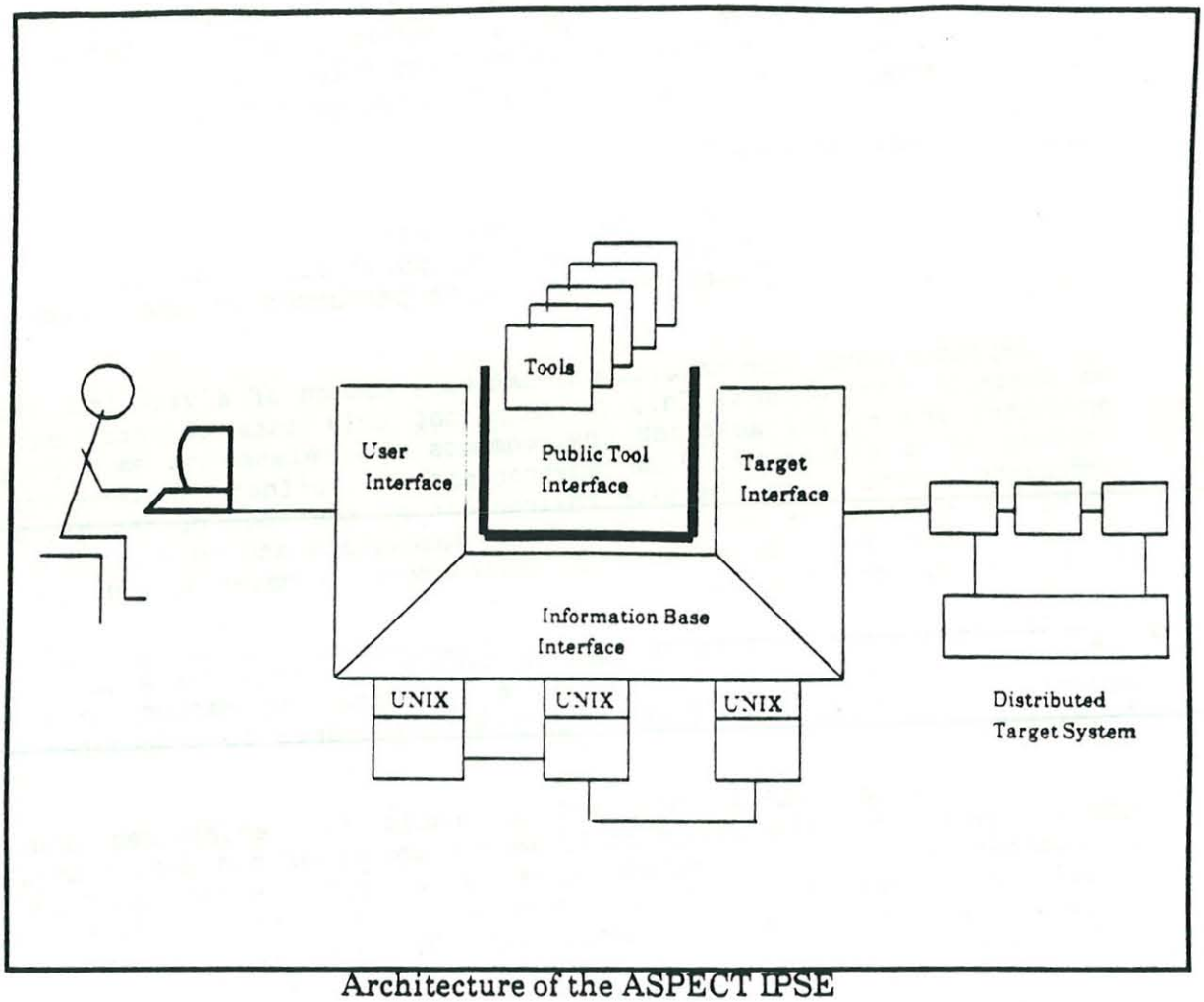
The structural architecture of Aspect consists of a set of kernel software components which cannot be by-passed. This kernel software is accessible via the Public Tool Interface. The kernel is built on top of the Unix operating system, and the open tool interface is seen as part of the public tool interface. It is not a direct route to the underlying Unix system. Distribution at the host level will be handled by the Aspect kernel and normally hidden from the tool writer. However, the PTI includes facilities to configure the system and to control distribution.

The functional architecture is shown in the following diagram. There are three main functional areas accessible accross the public tool interface. These are the Information Base, the User Interface, and the Target Interface. Aspect has the notion of activities which are analogous to transactions in a commercial database system. Activities are controlled and coordinated by an activity manager which provides the context within which tools run.

The Information Base

The information base is the central repository for all Aspect information. We have chosen to use Codd's extended relational model RM/T as the data model to provide the framework within which to store information. The Information Base also exhibits a variable granularity. In the software engineering environment the objects that are atomic to the information base may often be files. In Aspect, the value of a field in a record can be a pointer to a file.

The Information Base is also intimately connected with the process model. Activities are the unit of 'work' for the Aspect System and rules can be defined which control and check the progress of activities. Activities carry out their operations in the context of an abstract environment provided by a view of some subset of the information base.



Architecture of the ASPECT IPSE

The extended relational model was chosen principally because the classical relational model is a subset. This gives a well understood query language which besides being an essential part of an ipse also proved to be an invaluable tool during the construction of the information base. Using the relational model also meant that we could use a commercially available database system as the basis of our prototyping activities.

The relational model is extended in the sense that it captures more of the semantics of the data. RM/T identifies some relations as representing entities and their properties and other relations representing relationships or associations between entities. It enforces entity integrity, that there shall be no duplicate instances of an entity type, and referential integrity, that the entities referred to in an association must exist. There is a Catalog which describes the structure of the information base and this Catalog is itself held as relations thus allowing the same query language to be used. The model can also handle a sub-type hierarchy, something we consider to be important in a software engineering environment.

Objects in the information base are identified by surrogates. These are system generated entity identifiers, permanently associated with the

object and are never reused. The external naming is achieved via the 'known as' relation. This links a surrogate with its external name in the context of a name space. Name spaces themselves have surrogates, and this leads to a naming scheme analogous to Unix files. Entities need not have external names, but might be selected by their properties.

The Aspect Process Model

In the Aspect Process model, activities, rules and views form a coherent whole. Activities are the focal point for rule applications and provide the units of work that are to be performed in the context of a view.

Aspect extends the traditional database notion of a view into that of an Abstract Environment. This provides not only data objects, but also operators and rules. Abstract Environments are related to each other as members of a hierarchy. Each environment is defined in terms of its immediate parent, with the base environment at the root of the hierarchy. There are separate definitions for data, operations and rules, and a chain of definitions is followed when an environment is materialised. This may be interpreted or compiled.

The relational algebra is used to define data objects, with links to programming language procedures where required. Operations are also defined in terms of the algebra, again with recourse to procedures where necessary.

The rules governing data values should be centralised and not embedded in tools. This should minimise the amount of bad data that might be entered into the information base by a tool. Aspect provides the facilities for tool-writers to define rules in addition to the ones built in by virtue of using the RM/T data model. The relational algebra is used to define rules. False is associated with the empty relation and true with a non-empty relation. This gives the same expressive power as the first order predicate calculus and, because it is the same mechanism used for views and queries, gives great economy in implementation.

Rule application is tied to the activity structure. Executing activities form a hierarchy in an analogous way to nested transactions. These executing activities control the application of rules.

Rules may be divided into two types, those that relate to a particular state of the information base, and those that relate a before state to an after state. These are known as static and transition rules. A set of applicable static and transition rules is associated with each activity. These are inherited by the children of the activity and provide a context within which the children of the activity are carried out. The applicable sets are enforced on every state change.

Rules are also applied as pre-and post- conditions on every activity. These enable activities to be controlled on a project wide basis.

Basic operations are provided at the PTI to create and delete rules and to include or remove rules from applicable sets.

Activities define the limits of work to be carried out, they are, in general, long-lived and can span sessions. They are nested and form natural units for recovery. Activity definitions are objects held in the

information base. They can be used for the planning and coordination of a project. The pre-and post-conditions attached to activity definitions constrain the definitions and could prevent, for example, the execution of a quality assurance step if the previous step did not meet its post-conditions, for example, if the documentation was incomplete. Activity definitions are essentially descriptive, stating what has to be done, rather than exactly how it is to be carried out. The language used to define these activity schemes allows for the sequencing, iteration, refinement and choice of activities.

Executing activities carry out the transformation of the information base under the control of the pre-and post-conditions associated with the activity definitions and under the constraints of the appropriate sets of applicable rules. Executed and executing activities have a tree structure whose nodes are PTI primitives or calls to other tools. This execution tree records the past and current activities and can be used for a derivation history and for recovery.

Operations are provided at the PTI level to create/delete activity definitions and to associate them with rules. Executing activities can be created, started and completed.

Tools define and carry out sequences of Aspect operations, and can be viewed as extensions to the PTI. They can themselves invoke other tools and start/complete activities. Their invocation form part of the executing activity tree. At the PTI level tools can be invoked, complete execution or start and complete activities. Tools are added to the PTI by a process known as tool registration.

The Application Display Interface

The objectives of the application display interface, or adi, are to provide a simple intuitive model for the tool writer. The adi primitives should not constrain i/o style although individual ipses might develop a house style. There should be a maximum amount of display and input device independence and, as far as possible, generic input/output functions, such as editing, echoing, etc., should be handled by the system.

These objectives are realised in a system component called the presenter. This acts as a buffer between tools and the terminal. Display objects are considered to have a single logical structure. The presenter manages the mapping of this structure onto particular devices, and responds consistently to requests that came from tools or that came from the users of tools.

The logical structure of a graphics object is held as an ordered n-ary tree, whose leaf nodes are logical characters or graphical areas. The other nodes are regions with attributes, such as size, position etc., which determine the behaviour of the nodes below them in the tree. For example sizes are always relative to those of the parent region.

The primitive operations at the PTI are for the manipulation of the logical structure of the tree. There are operations to, for example, open a subtree, create a leaf, delete and cut. Other operations allow the attributes of regions to be queried and set. These attributes include sizeable - size may be changed in x,y direction, moveable - moveable in x,y direction, float - regions expand or contract to contain text.

A tool called 'double view' has been developed which allows one to see the logical tree and to query and changes its attributes and at the same time to see what effect this has on the actual displayed picture.

Target Interface

The target interface is not defined at the PTI as it was found difficult to define a general tool set that would support all architectures. Instead interfaces will be published for particular tool sets to allow the additions of further tools to the family.

Distribution

The objective of distribution is to provide transparent access to data at the logical level. In general, tool writers should not have to know about distribution. There is however the need to define a model for describing the physical system. Objects are associated with replicas which are stored on volumes mounted on peripherals. Peripherals and users are associated with devices which are interconnected by channels. The logical, undistributed model is mapped onto the physical model.

Formal Definition

The Aspect public tool interface was formally defined in Z, a language developed by the Programming Research Group in Oxford. This had several benefits. It gave the project a common technical language within which to discuss alternative designs. Writing the Z definitions cleared up many loose ends at an early stage and made subsequent implementation much easier. From the point of view of industrial collaboration a detailed definition is probably more portable than prototype code.

Implementation

The first phase of the Aspect implementation was completed in December 1986. This included the application display interface, the underlying RM/T database engine and the rules and views part of the information base superstructure. The next phase of the implementation will integrate these ideas using the activity mechanism and also incorporate concurrency.

DISCUSSION

Dr. Ritchie wanted to know more about the way the Aspect Kernel would handle Unix calls.

Although Unix would be used to carry out Unix calls, Aspect does not allow this to happen in an uncontrolled way. To provide such control we can think of the Aspect Kernel providing a harness within which Unix tools can be registered and run. All Unix calls will be trapped by, and files provided via this harness before allowing Unix to execute the call.

Dr. Harrison speculated that, if the tools

(a) for deriving information;
(b) for doing rules checking;
(c) for providing the user interface;

were all provided by the System this perhaps changed our perception of what tools were designed in a development environment.

Dr. Hitchcock agreed.

DISCUSSION

Professor Whitfield asked to what extent will the history of the tree of executing activies be recorded.

Dr. Hitchcock replied that in principle the whole history would be recorded although for implementation and storage purposes this may need to be selective.

Mr. Jackson asked whether views can be used to restrict access to tools only to authorised users?

Dr. Hitchcock stated that the answer was yes. Views have three parts, data, operations and rules. Tools are part of the operations part.

Professor Brown was puzzled as to why the coordinate system associated with the presenter works on reals between 0 and 1.

Dr. Hitchcock said that the reason is because the size and position of each structure in the logical model (a tree) used by the presenter is defined relative to its parent.

Dr. Harrison wondered whether this relative scaling did not imply that the children of certain nodes may be made to disappear altogether.

Dr. Hitchcock said this was possible but that it was also possible to build in constraints to avoid the problem.

Professor Colouris asked whether the presenter was linked to objects in the IB so that the contents may be immediately displayed.

Dr. Hitchcock stated that the answer was no. The presenter is autonomous. A tool might read values from the IB and display them under the control of the presenter. Also the presenter itself will use the IB to hold its own tables etc.

Dr. Good asked who changes the attributes in the nodes of the presenter's logical tree.

Dr. Hitchcock answered that the attributes may be changed by a tool or by a user, under the direction of a tool.

Dr. Harrison asked about the 'Lock' attribute associated with the presenter - how can it be changed without the help of the application program.

Dr. Hitchcock said that changes will be monitored by the application and, if suitable filtering were used, the lock command would be unnecessary. However it is there as a convenience and enables trees to be stored away or used in another context.

Professor Randell asked what the present status of the HCI work is.

Dr. Hitchcock answered that the HCI part of ASPECT was the most advanced as far as implementation was concerned. The situation however was complicated by the fact that the project is just changing to the Sun workstation.

Professor Whitfield asked for more elaboration in regard to the Target interface, in particular why it was not possible to generalise on the target architecture.

Dr. Hitchcock stated that the reason was basically because the architecture in generalisation did not concern chip architecture as such but rather network (bus) architecture.