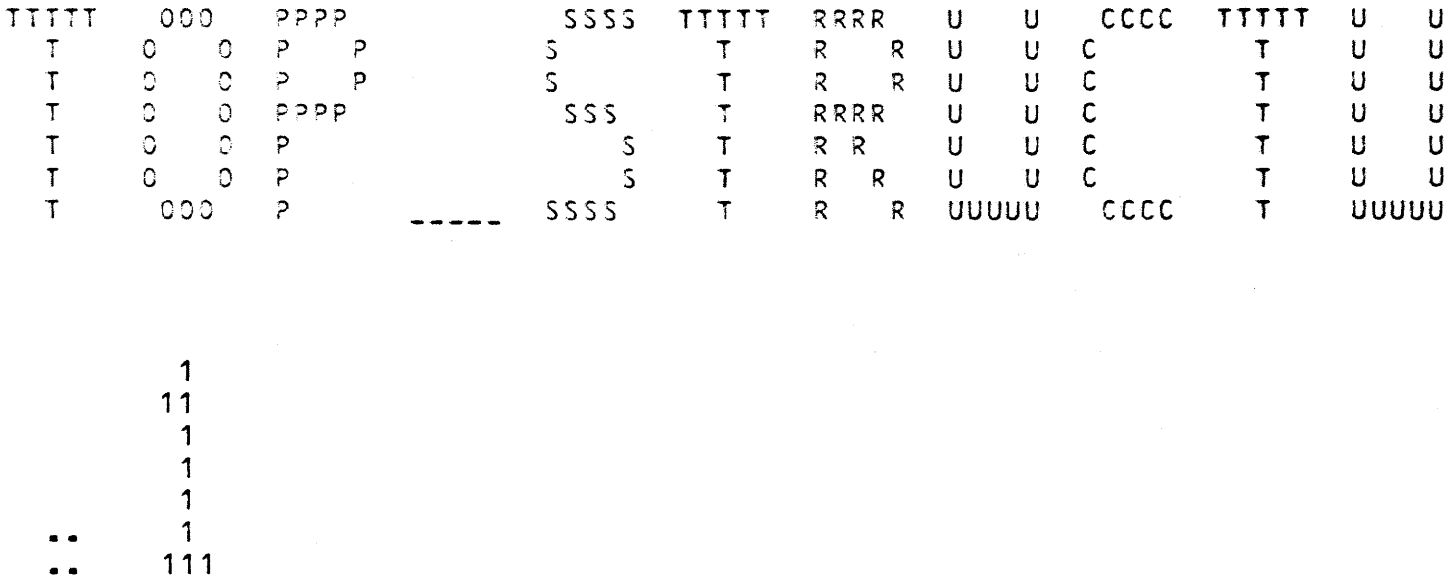

START Job TOP_ST Req #170 for EGB Date 28-Sep-82 22:06:32 Monitor: Rational
File RM:<RPE.DOC>TOP_STRUCTURE..1, created: 19-Mar-82 22:31:23
printed: 28-Sep-82 22:06:33
Job parameters: Request created:28-Sep-82 22:06:23 Page limit:27 Forms:NORMA
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:

Top Level Structure

A - user view

The user sees the following Ada package. Semantics of all operations is explainable from this package except for a limited number of "meta" operations.

```
package body R1000 is

    -- misc. utilities and things reside here
    package RawIO is ... end RawIO;
    -- etc.

    Package Edit is
        procedure move_up;
        procedure delete;
        -- ....
    end Editor;

    package Debug is
        procedure set_break;
        procedure reset_break;
        procedure trace;
        -- ....
    end Debug;

    package TEXT is
        type entity is (word, line, sentence, paragraph, page, document);
        package edit is
             -- defines basic text operations
            procedure justify(what: entity:= paragraph);
            procedure fill(what: entity:= paragraph);
            -- ...
        end edit;
    end TEXT;

    package program is
        package edit is
            procedure find_decl;
            procedure fully_qualify;
            procedure start(t: diana.tree);
        end edit;
    end program;

    package tty_io is
        -- note, in this schema we can use the package as in the manuasl
        procedure put(c: character);
        procedure put(s: string);
        -- ....
    end tty_io;

    -- other object editors include Status, Management system, Mail

    package SITE is
        -- organization within package site is completely up to the
        -- installation
        package GROUP1 is
            package USER1 is ... end USER1;
```

```
            package USER2 is ... end USER2;
        end GROUP1;
    end SITE;

    package body SITE is
        package body GROUP1 is
            package body USER1 is
                ...
            end USER1;

            package body USER2 is ... end USER2;
        end GROUP1;
    end SITE;

    -- bodies of all system package reside here, invisible to users.

end R1000;
```

Operations:

Each user has a default package somewhere within Site. The exact position of visible part and body are determined by the site management.

After logon to the system the PE will create an editor and a window displaying the user's default package. There will be the option to save the state of a sessions; after login the state of a previous session my be restored and this session continued.

All user communication with the system will be done through the editor unless the user performs basic IO operations (This will be made fairly hard to do and will not be the normal case).

The user may move his edit window to other points within R1000 according to Ada visibility and access rights as described below.

Commands are Ada procedure calls. All commands can be explained according to Ada semantics in the current context (see below).

Commands may be bound to keys or key sequences; typing explicit commands is aided by command completion and systax directed editing. Single keys and key sequences may be echoed in verbose mode in their full Ada syntax.

According to the current mode the system will provide appropriate prompts indicating the current status to the user. E.g.
>
will be the prompt for arbitrary Ada statements
>Edit.
will be the prompt for commands that will be directed to the Editor.
>Text.Edit.
will be the prompt for the text object editor.

Meta Operations

For the system to work certain operations outside Ada have to be allowed. In a strict sense the execution of Ada statements in a given context is such a meta operation since Ada does not allow dynamic execution of statements.

In addition there are two meta operations; elaboration and unelaboration of pieces of Ada programs. These operations transform program instances to program text (diana trees) and vica versa; elaboration may create a new version of a program.

The existence of multiple running versions of the same program can be explained completely on the source level; i.e. the old version and all its uses will be consistently renamed and a new version will be added without introducing inconsistencies.

Access Rights

The environment will allow users various forms of access to components of the system.

1) CONTEXT ACCESS to a module gives the user the right to view this module (visible part and body) with the editor and execute statements in the context of this package; i.e. statements are interpreted as if they were declared in this module. Context access implies the right to debug tasks. Context access implies read access (see below) to all entities visible from the context according to Ada scope rules.

2) WRITE ACCESS to a module implies context access to the module. In addition write access grants the right to (a) unelaborate components of the accessed module, (b) create new version of components of the accessed module, and (c) create and elaborate new components withing the accessed module.

3) READ ACCESS to a module allows to read the program text of the module. Read access does not imply context access, e.g. a user may be granted read access to the body of a package but (s)he may not be allowed to execute within the context.

Rationale: Read access to a package means read access to the program text of the package in the traditional sense. The user may inspect and copy the text of the program but may not modify the original copy. Context access means access to a program instance and the associated environment. Variable values of this environment may be inspected and altered but the program instance itself may not be changed. Write access is the highes access right, it allows complete modification of the program text and the program instance.

8 - Implementer's View

```
package body R1000 is

    Package Edit is
        -- ....
    end Editor;

    package Debug is
        -- ....
    end Debug;

    package TEXT is
        -- ....
    end TEXT;

    package program is
        -- ....
    end program;

    package tty_io is
        -- ....
    end tty_io;

    package SITE is
        -- ....
    end SITE;

    package body SITE is
        -- ....
    end SITE;

-- apse related things

    package elaborator is
        procedure execute(user: stack_name; statement: diana.tree;
                          current_terminal: window_control_block);
        procedure elaborate(user: stack_name; declaration: diana.tree;
                          position: diana.tree;
                          current_terminal: window_control_block);
    end elaborator;

    task type editor is
        entry move_up;
        entry delete;
        -- ....
    end editor;

    type editor_name is access editor;

    type window_control_block is
        record
            core_editor: editor_name;
            Ada_editor : ada.editor_name;
            window     : window.id;
        end record;

    task session_manager is
        entry assign(user: stack_name; terminal: window_control_block);
        entry release(user: stack_name);
        entry retrieve(user: stack_name; found: out boolean;
```

```
                    terminal: out window_control_block);
end session_manager;

Package body Edit is
    procedure move_up is
        found: boolean;
        terminal: window_control_block;
    begin
        session_manager.retrieve(system.myself(), found, terminal);
        if not found then
            raise no_terminal_assigned;
        else
            terminal.core_editor.move_up;
        end if;
    end move_up;

    procedure delete is
        -- ....
    begin
        -- ....
        null;
    end delete;

    -- ....
end Editor;

package body Debug is
    -- .... similar basic strategy as editor ....
end Debug;

package body TEXT is
    -- ....
end TEXT;

package body program is
    --
end program;

package body tty_io is
    -- ....
end tty_io;

package body elaborator is
    procedure execute(user: stack_name; statement: diana.tree;
                      current_terminal: window_control_block) is
    begin
        -- generate the proper code segement etc.
        session_manager.assign(user, current_terminal);
        user.continue(new_code_segment);
        session_manager.release(user);
    exception
        when others =>
            current_window.report_error("unhandled exception");
            session_manager.release(user);
    end execute;

    procedure elaborate(user: stack_name; declaration: diana.tree;
                        position: diana.tree;
                        current_terminal: window_control_block) is
    begin
        if write_access(user) then
            null;
            -- .... similar to execute ....
```

```
            else
                current_terminal.report_error("write access required");
            end if;
        end elaborate;
    end elaborator;

    -- etc .....

end R1000;
```

Operations:

The system keeps a table associating user names with passwords, default packages, and existing sessions of a user. For the default package the system keeps a stack name (capability) which is used to grant write and context access to this package.

There will be different versions (or modes) of Ada object editors. A basic version merely understands program text while other versions understand program instances. One of these object editors is the Debugger.

Any command typed to the bare prompt ">" will be interpreted as Ada statement in the current context and executed if valid.

Any command typed after the ">Edit." prompt will be known to be a command to the editor. Such a command need not be executed by elaboration, rather the appropriate function can be invoked directly. In case the user redefines the package Edit the system will change the normal edit prompt to ">R1000.Edit.".

Similarly, commands to object editors can be shortcircuited.

Meta Operations

To elaborate, i.e. run code on a user stack the system will enter the user stack name together with a description of the current terminal (editor tasks) in a data base (session_manager).

Calls to visible procedures of edit packages (e.g. debug, edit, program.edit tty_io etc) will look up the required terminal description from the data base and route calls to the appropriate edit task.

Access Rights

To get read access an editor merely needs access to the diana representation of the object.

Context access requires the stack name (capability) of a package instance. The distinction between context and write access is enforced by the software; the distinction between context and read access is enforced by the hardware (capabilities cannot be created)