A Technical Introduction to the Think Framework

Jean-Charles Tournier
12/06/2005
Outline

• Principles
• How Think works?
  – Overview
  – Implementation
  – Directories structure
• How to work with Think?
  – ADL/IDL
  – Components source code structure
  – Event prototype
Principles

- Objective: build dedicated OS kernels
- No pre-defined kernel philosophy (e.g. Mono, micro, exo, etc.)
- No pre-defined core functionalities
- Implementation of the Fractal component model
  - Concepts of: component (primitive/composite), controller, interfaces (type+name), bindings
  - Hierarchical model
  - Components are run-time entities
How does Think work?
Overview
Implementation

- Tools are written in JAVA
- Components are written in C and assembly
- Compilation tool chain is using ant
- Defines Vtables as C structures
interface Test {
    int bar(int arg);
    int foo();
} RTest;

typedef struct {
    struct Mtest* meth;
    void *selfdata;
} RTest;

struct MTest {
    jint (*bar)(void* this, jint argl);
    jint (*foo)(void* this);
};
Implementation – Using vtables
Directory structure

- **Build/**: tool chain binaries
- **Config/**: main ant file + definition of configuration properties (compiler options, compiler names)
- **Doc/**: doc.... But don't except too much! Some of them are not updated since the first version...
- **Src/**: source code of the component library.
- **Example/**: kernel examples. Most of them are dedicated to iPaq.
- **Tools/**: ADL/IDL parser, glue code generation, etc.
How to use Think?
- The HelloWorld example-
Global design
Architecture Description Language

type MainType {
    provides activity.api.Main as main
}

type hwType extends MainType {
    requires video.api.Console as console
    attributes position
}

primitive hwmain implements hwType {
    skeleton hw nolifecycle
}

primitive hwmain {
    requires video.api.Console as console
    provides activity.api.Main as main
    attributes position
    skeleton hw nolifecycle
}
composite hwBase implements RootType {
    contains console = arm.sa1100.h3600.video.lib.screen
    contains hw = hwmain
    binds this.main to hw.main
    binds hw.console to console.console
    assigns hw.position = 3
    controller org.objectweb.think.controller.Static
}
package activity.api;
interface Main {
    void main(int argc,
                string[] argv);
}

package video.api;
interface Console {
    void putc(char c);
    void putcs(string str);
    void putxys(int x, int y,
                 string str);
    void scrollup();
    int cols();
    int rows();
}
Hello World source code

```c
#include <activity/api/Main.idl.h>
#include <video/api/Console.idl.h>

struct hwmaindata {
    // Imported interfaces
    Rvideo_api_Console* console;
    int position;
};

#if ! defined(ONLYDEFINITION)

/* Main interface method */
static void mainentry(void* _this, jint argc, char** argv) {
    struct hwmaindata* self = (struct hwmaindata*)_this;
    CALL(self->console, putxycs,
         self->position,
         self->position, "Hello World!");
}

struct Mactivity_api_Main hwmain_mainmeth = {
    main: mainentry
};
#endif
```
Event prototype

• Implementation of 2 consumers and 2 producers for a single evt E.
• Based on the mediator pattern
• Technical issues:
  – No dynamic interface creation => a required itf can now have a set of values (list) rather than a single one
• Next:
  – Automatic event binding => modify ADL
    • e.g: \texttt{eventbinds cons1.e to mediator.e}
    • Semantic of such binding?