Architecture, paradigms, idioms and weirdness of the C++ in your pocket!

John Pagonis

Senior Software Engineer / Senior Developer Consultant

Developer Consulting

Symbian Software Ltd

symbian

Welcome :-)

- This talk is about the Symbian OS C++ framework
- For people interested in knowing the “Whys”
  - … behind some Symbian OS idioms
  - … behind Symbian OS C++ dialect
  - … mobile development
- For people that like to “hack” below the surface
- For people familiar with (Symbian) OS, frameworks, language and architecture.
- Why things are the way they are…..
- …and remember “Interrupts, help unblocking’’

Why this talk

- Because Symbian OS and Symbian OS C++ are different
- Because it is interesting
- Because people may be tempted not to spend time to appreciate it :-/
- Because of it’s mass adoption and deployment we run a risk of badly educated engineering…
- Software archaeology teaches us a lot

- “Never criticise what you don’t understand”

Today’s menu

- Some context
- Intro to the OS
- About language selection
- Symbian OS C++ idioms (a.k.a. weirdness for a reason ;-)

To bootstrap …

- By Q406 there were 110 million Symbian OS phones!
- Q406: 14.6 million shipped
- There have been 157 different models on the market
- By Sony Ericsson, Nokia, Motorola, Samsung, Sharp, LG, Fujitsu, BenQ, Siemens...
- All phones use one of 3 different GUI platforms
- The cost of making a first of a generation mobile can easily reach 40 million USD and take more than 2 years for an experienced manufacturer.
- 2 Symbian OS smartphones shipping every second!

About architecture…

- Every architectural feat exists within some context, where it evolves under (competing many times) forces.
- That context and such forces present the architect with constraints and affordances.
- For us, ‘’small mobile personal computers’’ has always been that context.
- We’ve been operating in this context for the past 20+ years or so !

Some historical background

- Early 80s: Spectrum ZX/81 Flight Simulator and other games done by Psion
- 80s : 8bit Organiser circa ‘84 (I, II, P350, CM, LZ64 etc) and the Sinclair QL app suite – early frameworks
- Late 80s : SIBO/Epoc16 circa ’88 – WIMP(MC), HWIM(s3), XWIM(s3a) - emerging frameworks
- 90s : Epoc32 – EKA1, HCIL, Eikon – frameworks
- Late 90s – today: Symbian OS – EKA1/EKA2, Uikon, UIQ, S60, MOAP(FOMA) UI etc

Evolution…

- Games (efficiency and hardware insight)
- Vertically integrated products (hardware and software), OS and app development
- Product diversification, Frameworks, OS creation and evolution (many of)
- Software reuse and platformisation

- Z80 assembler
- x86 assembler, Forth, OPL, C, systems thinking, UI, first object-based designs with C, limited comms, early OOP
- OOP/OOD, C++, Java, OPL, comms
- J2ME, Python, Ruby, C++, many GUI frameworks, lots of comms, open to anything and anyone really…

Some of the books that influenced us..

- “The Psychology of Computer Programming”, G. M. Weinberg, 1971
- “The Mythical Man Month”, F.P. Brooks, 1974
- “Object Oriented Programming: An Evolutionary Approach”, B. C. Cox, 1986
- “Object Oriented Software Construction”, Bertrand Meyer, 1988
- The C++ Programming Language”, Bjarne Stroustrup (gray version), 1991
- “Advanced C++ Programming Styles and Idioms”, J. O Coplien, 1991
- “Effective C++: 50 Simple Ways to Improe your software and Designs”, Scott Meyers, 1992
- “Taligent’s Guide to Designing Programs”, 1994
- “Design Patterns: elements of reusable object-oriented software”, E. Gamma et al., 1995

Question:

- How many lines of code are there in a Symbian OS smartphone today?

a) approx. 250 KLOC

b) approx. 500 KLOC

c) approx. 1 MLOC

d) approx. 5 MLOC

e) approx. 10 MLOC

f) approx. 15 MLOC

g) Too scared to find out :-)

Some Greek definitions (John style)

- Architecture == discipline/way of building a construct
- Architect == lead constructor (carpenter in ancient Greek)
- ‘deigma’ == {sample, exhibit}
- ‘paradeigma’ == example of …
- Paradigm == a way of doing/showing something approximate
- ‘Idios’ == self
- Idiom(a) == native dialect (way of using, expressing a language and/or its vocabulary), special characterising qualities, peculiar habit characterising a person

Note: ‘idios’ and ‘idiot’ are only one character away :-)

Language selection… C++

- C++ is a multi-paradigm programming language
- Probably not the best language for loosely coupled open world systems
- But we pretty much knew what the apps would need and frameworks should be like
- It supports OOP
- Good for low level OS construction, strongly typed and works nicely with assembler
- Has too many features and demands proper attention, only got standardised recently…

How did we select C++?

- Well, Psion Software in the late 80's was using an Objective-C look-alike called POOC
- Bjarne Stroustrup was giving a lecture at Imperial College in London at the time
- David Wood was attending
- Graduates were getting hired in th early nineties to work on a new ''background project''
- Graduates didn't want to learn a proprietary language
- Objective-C was going nowhere at the time
- C++ was getting more traction (compilers etc)
- …the rest is history as they say

Symbian OS C++

We refer to Symbian OS C++ as the C++ **domain specific *dialect*** and accompanying frameworks that we use to build Symbian OS and the software that runs on it.

- This dialect as well as the OS itself have emerged from the vast experience dealing with small mobile computers.
- This experience has been captured in the domain specific idioms and paradigms of Symbian OS.
- **Some of the idioms and paradigms present in Symbian OS are reflected in Symbian OS C++ and vice versa.**

Wot no Standard C++ ???

Domain forces, late standardisation, compiler conformance and compiler performance meant that :

- We had to came up with our own exceptions handling mechanisms
- Came up with our own collections, container and buffer/string handling frameworks
- Concurrency support was not in the package

…and we have been criticised ever since, for architecting something different that works well and is domain specific :-(

Symbian's context: "Small mobile personal computers"

- User Interaction and Responsiveness
- Battery Powered
- ''Always-on'' , always pocket-able
- Reliability
- Resource constrained
- Openness…

Some paradigms…

In order to support the user-centric mentality and operate within the mentioned forces we used architectural paradigms and invented idioms such as :

- Multithreading and pre-emptive multitasking
- Lightweight micro-kernel OS design
- Client-Server, session based IPC (among other mechanisms)
- Asynchronous services, Active Objects
- Cleanup Stack, Leaves, Traps for exception handling
- Re-usable frameworks for apps, middleware, GUIs
- …and descriptors

Micro-kernel design

- Lightweight micro-kernel with client server architecture
- System servers as well as user servers run in user space
- Kernel uses a Virtual Memory Model and MMU for memory protection
- Drivers run kernel side
- Client/server session based IPC for server access
- Servers mediate access to shared resources and services

EKA2 (Epoc32 Kernel Architecture 2)

- Multi-threaded, pre-emptible Real Time Kernel
- In-fact is a Symbian OS personality on top of a nano-kernel – **many** personalities can co-exist ;-)
- O(1) scheduler
- System calls are all pre-emptible as well → dual stacks
- Deterministic ISR, thread response, latencies etc
- Memory models and local allocator strategies can be plugged-in
- Many more IPC/ITC mechanisms such as local/global message queues, publish-subscribe, global anonymous queues, shared and global memory chunks

Kernel – Nano-kernel and personalities

- EKA2 splits the kernel in two layers
- A nano-kernel and a Symbian OS kernel
- The Symbian OS kernel is a micro-kernel
- The Symbian OS kernel is a ‘’personality’’ on top of the NK
- There can be many such personalities running simultaneously, thus many micro-kernels can run pre-emptively on top of the Symbian nano-kernel !!!!!!
  - For example, one for the GSM or UMTS stack and one for Symbian OS
- NK is responsible for the very basic synchronisation, timing, initial interrupt handling and scheduling services
- It is not depended on the system library and doesn’t know about processes or memory models (dealing with the MMU or lack of)
- All offered services are deterministic with bounded execution times

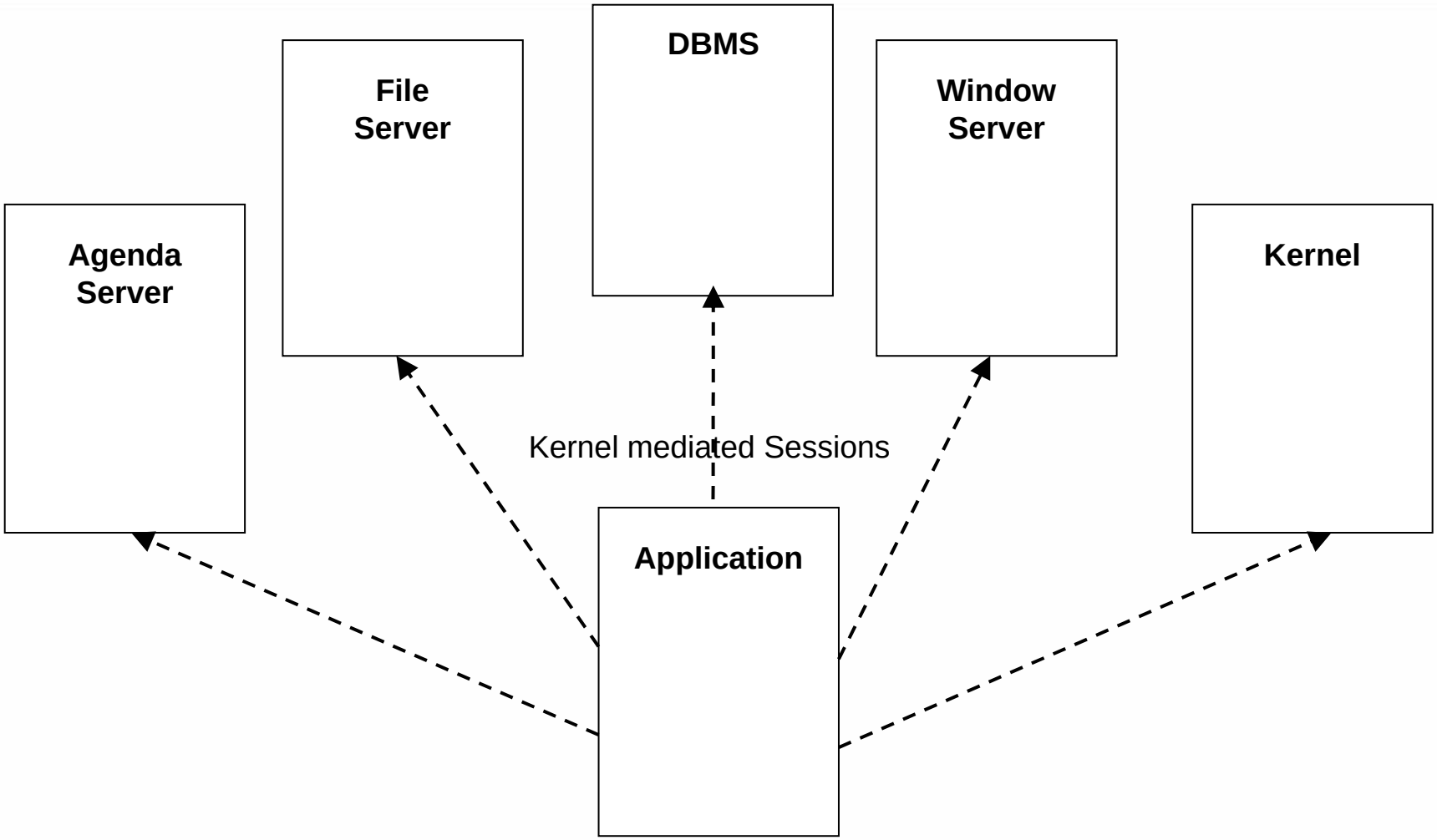
Kernel pre-emptibility

- The EKA2 Symbian OS Kernel is multi-threaded
  - … five threads and an executive (collection of system calls)
- It is completely pre-emptible
  - … All memory allocations and even a context switch can be pre-empted
- User side threads have a user mode and supervisor mode stack
  - … executive calls run on user thread's supervisor stack
  - … executive calls can all be pre-empted!!!
  - … executive calls can be installed per thread, even!!!

Four Symbian OS C++ idioms

- Active Objects
- CBase and other type idioms
- Descriptors and run time bounds checking
- Exceptions handling

Asynchronous services



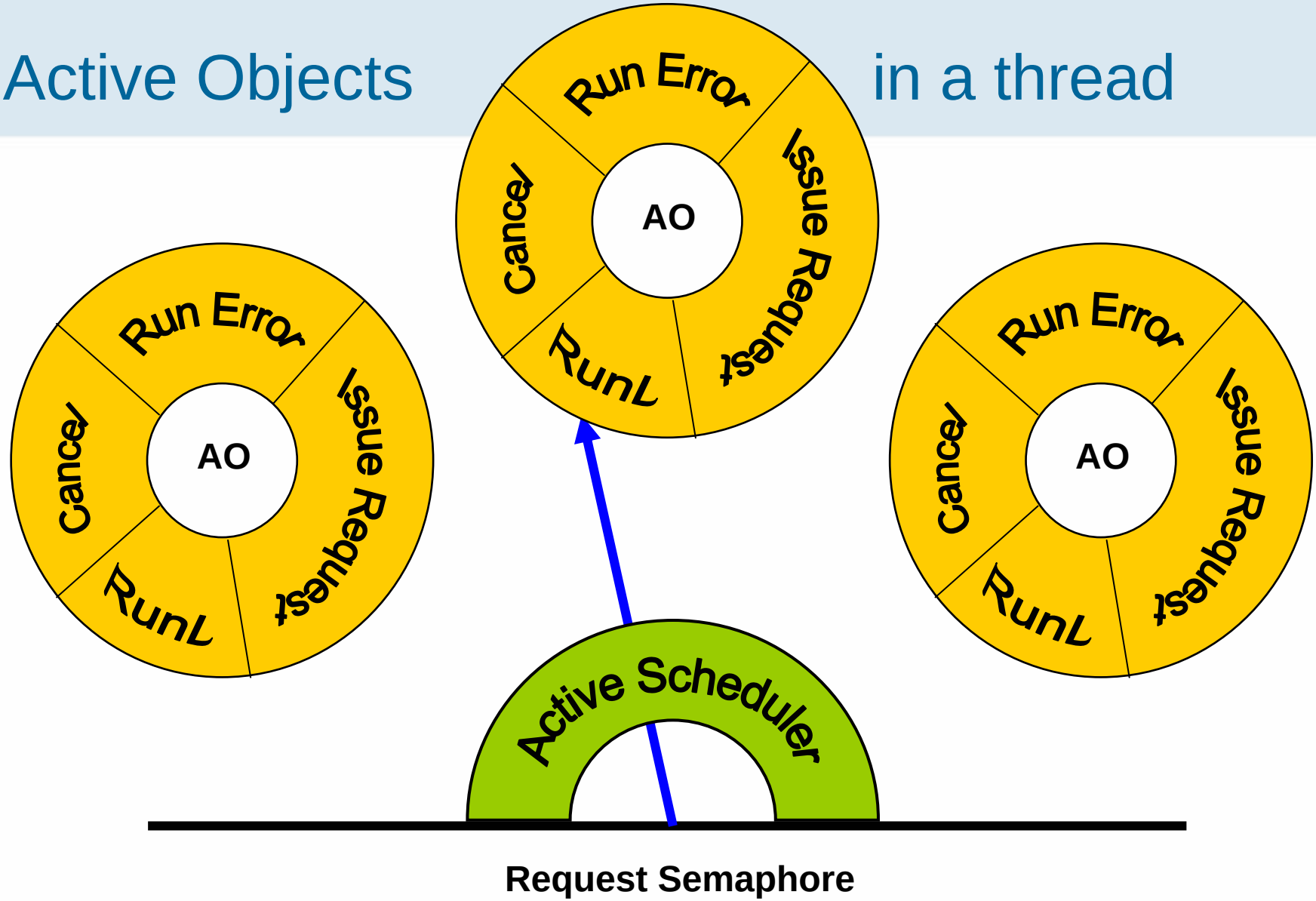
Many asynchronous services….

- But the OOP model is **serial** and so is vanilla C++
- Concurrency support was needed for asynchronous event handling and the multitude of client-server interactions
- A lightweight model that introduces concurrency and **avoids synchronisation issues** is that of Active Objects
- btw: Grady Booch talks about AOs in his 1990 book on ‘’Object Oriented Design with Applications’’. Defined as objects having their own thread of control.
- We used collaborative AOs within a thread instead based on the late 80’s and SIBO experience.

Active Objects

- Thus every thread got a *request semaphore*
- Most threads and certainly all apps and servers got an *Active Scheduler*
- So that it can schedule…. Active Objects
- Thus Active Objects became a lightweight mechanism for **encapsulation of concurrent transactions** over session based IPC
- Simple: Issue a request, run when the request completes, if an error occurs handle the error.
- … priorities, run to completion, no pre-emption

Active Objects in a thread



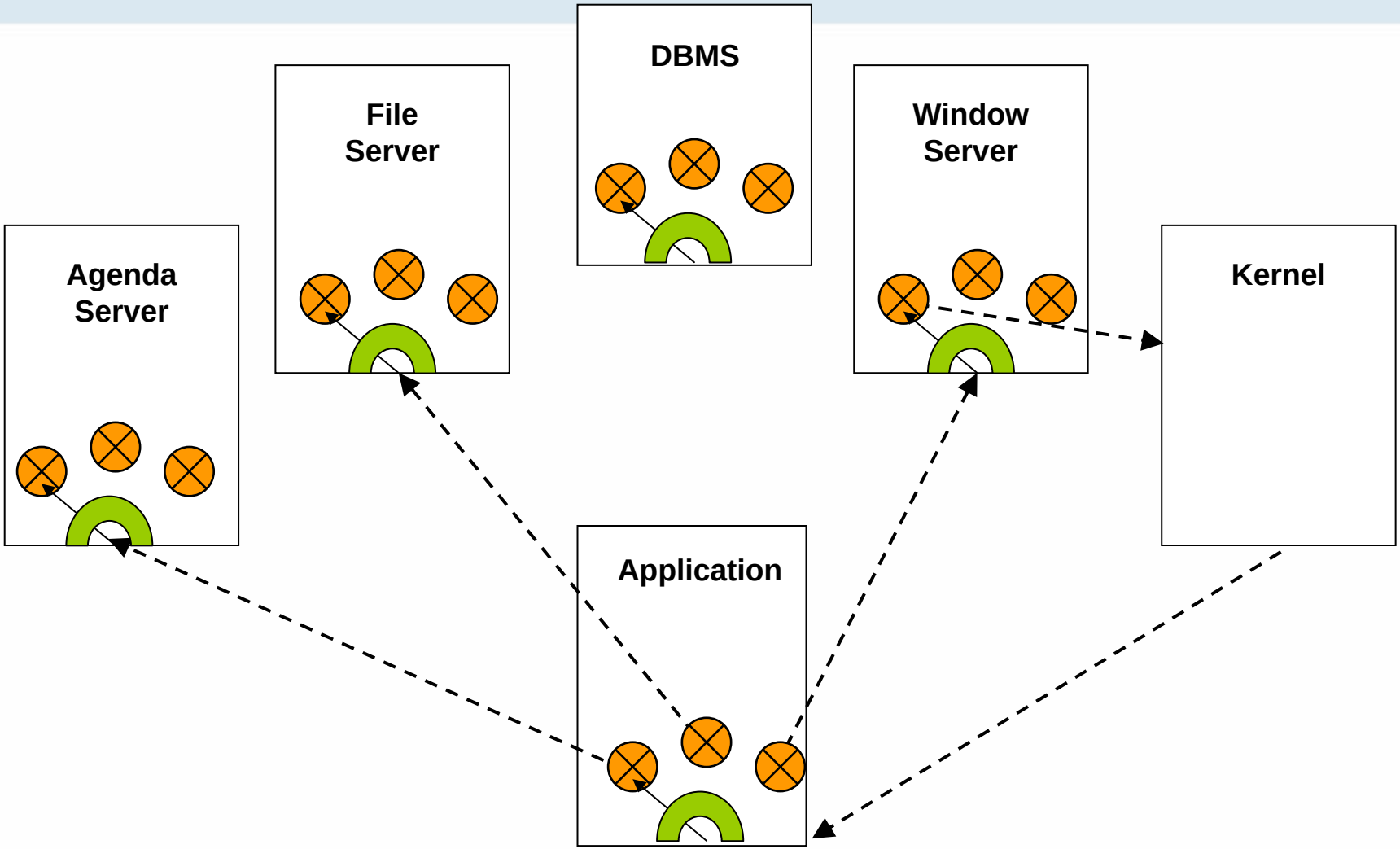
The Active Objects interface

```
class CActive : public CBase
    {
public:
    IMPORT_C ~CActive();
    IMPORT_C void Cancel();
    IMPORT_C void Deque();
    IMPORT_C void SetPriority(TInt aPriority);
    inline TBool IsActive() const;
    inline TBool IsAdded() const;
    inline TInt Priority() const;
protected:
    IMPORT_C CActive(TInt aPriority);
    IMPORT_C void SetActive();
    virtual void DoCancel() =0;
    virtual void RunL() =0;
    IMPORT_C virtual TInt RunError(TInt aError);
public:
   TRequestStatus iStatus;
    }
```

Active Object transaction encapsulation

- Use or open a session to some service provider (server)
- Add AO to the Active Scheduler
- Issue a request to that service that may complete asynchronously
- Set the AO active, thus notify the scheduler
- … handle the completion of the request in the `RunL`
- If there is an unhandled exception `RunError`
- ..or let the Active Scheduler know

Active Objects almost everywhere



Good things about Active Objects

- AOs help to manage complexity and synchronisation issues with concurrency within a thread, without too much to worry about
- AOs encapsulate transactions and their error handling
- AOs help us avoid multi-threading for async event handling where it is not always needed or when it is more complicated or just an overkill
- AOs let servers manage many clients and many transactions with just one thread

..and the bad

- BUT they are **no panacea** and should not be used where they don’t fit the paradigm !!!
- Multi-threading instead of AOs is valid and must be used where it must !

From Bjarne Stroustrup's FAQ:

Why doesn't C++ have a universal class Object?

We don't need one: generic programming provides statically type safe alternatives in most cases. Other cases are handled using multiple inheritance.

There is no useful universal class: a truly universal carries no semantics of its own.

A "universal" class encourages sloppy thinking about types and interfaces and leads to excess run-time checking.

Using a universal base class implies cost: Objects must be heap-allocated to be polymorphic; that implies memory and access cost. Heap objects don't naturally support copy semantics. Heap objects don't support simple scoped behaviour (which complicates resource management). A universal base class encourages use of dynamic_cast and other run-time checking

CBase

- All **C**omplex objects in Symbian OS must inherit from `CBase`
- Unless they are proxies to some other **R**esource
- Or they are simple **T**ypes that do not own any heap allocated objects.
- `CBase` gives you overloaded `new` that uses Symbian OS exceptions and cleanup mechanisms
- Zeroes out all its members
- Has virtual destructors and constructors
- By convention forbids multiple inheritance (we use mixins instead)
- Needs to be allocated on the heap…

CBase

Circa late 80's, during development of SIBO/Epoc16 using POOC (Psion Object Oriented C). It was observed that experienced programmers would zero init their members immediately following construction.

Therefore the allocator for 'classes' was handcrafted to do this and thus make complex object construction more efficient and safer for the careless programmers.

Later in '94, Epoc32 C++ inherited this.

Following this CBase inherited some features to go with how Symbian OS does exceptions handling and cleanup…

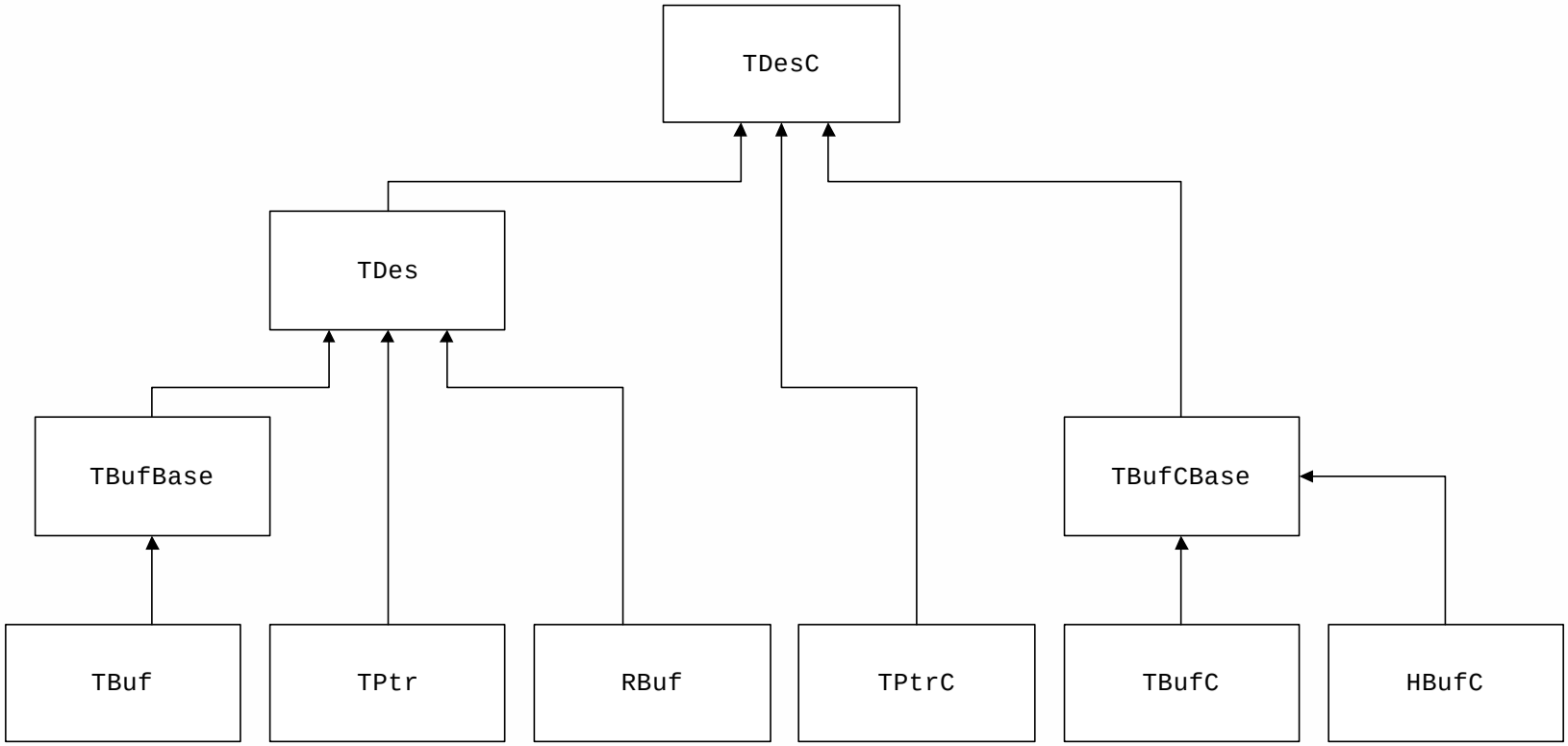
Descriptors not strings!

- Symbian OS C++ makes use of objects that encapsulate buffers or pointers to buffers as opposed to naked ‘C’ strings
- Descriptors are everywhere in Symbian OS
- Descriptors **do run-time bounds checking**
- Descriptors are ROMable
- Descriptors can describe buffers of ASCII, Unicode and binary data on the heap or on the stack
- They offer a safe and consistent mechanism for dealing not only with strings but also with binary data

Descriptor classes

- All descriptor classes inherit from the base class `TDesC`
- There are four ‘abstract’ classes: `TDesC`, `TDes`, `TBufBase` and `TBufCBase` (TBufCBase and TBufBase are an implementation convenience)
- These are not abstract in the C++ sense as descriptors do not have pure virtual methods; however they are abstract in the sense that they are not intended to be instantiated.
- There are six concrete descriptor types: `TBuf`, `TPtr`, `RBuf`, `TPtrC`, `TBufC` and `HBufC`
- In addition there is a related type known as a ‘literal’.‘Lits’ are not really descriptors but they are closely related.

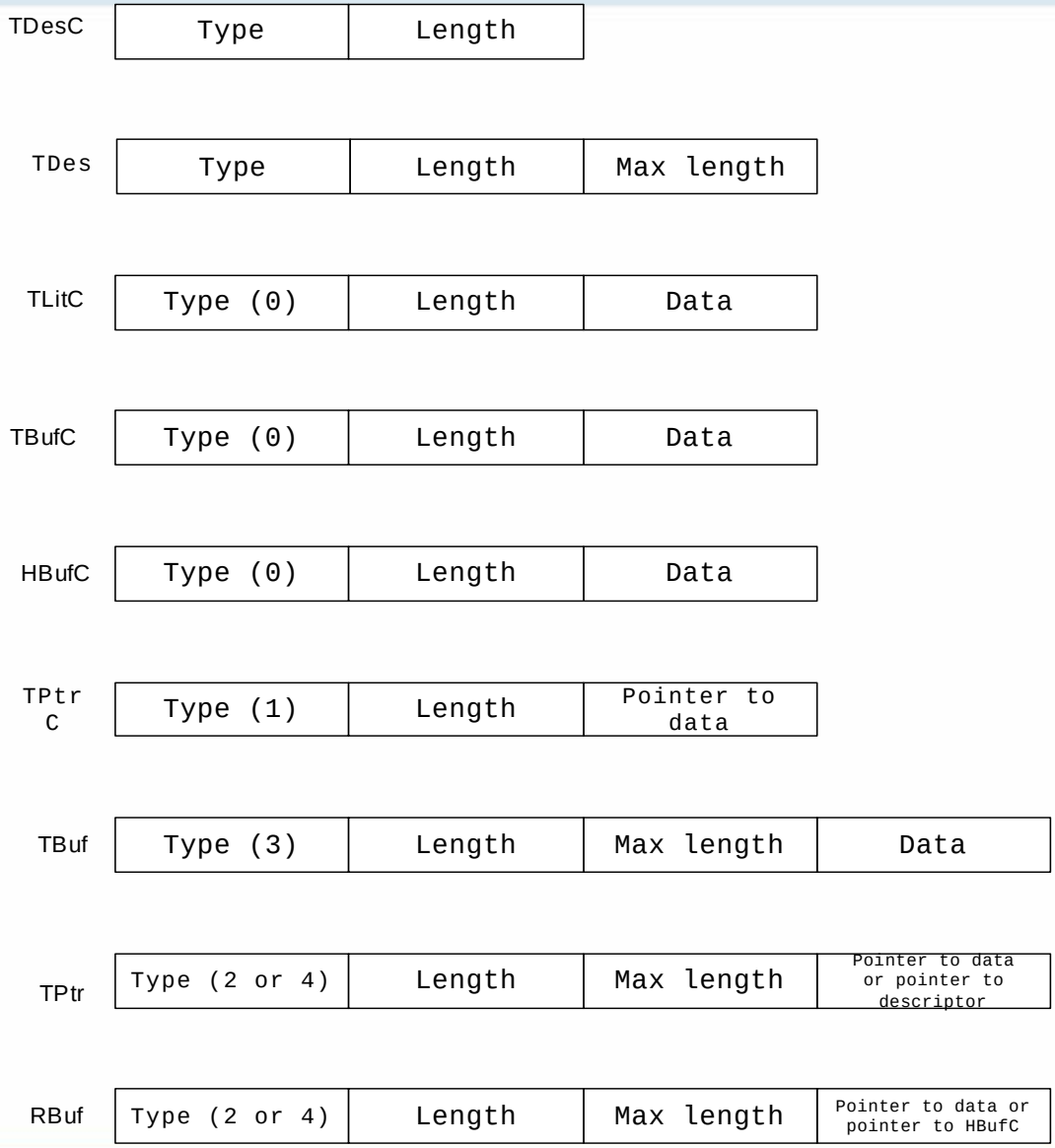
Descriptors hierarchy



Descriptor attributes

Type	Constness	Name	Approximate C equivalent
Literal	**Not modifiable**	TLitC8	static const char[]
Stack Descriptor	**Modifiable**	TBuf8	char[]
Stack Descriptor	**Not directly modifiable**	TBufC8	const char[]
Pointer Descriptor	**Modifiable**	TPtr8	char* (pointing to non-owned data)
Pointer Descriptor	**Not modifiable**	TPtrC8	const char* (pointing to non-owned data)
Heap Descriptor	**Modifiable**	RBuf	char* (pointing to owned data
Heap Descriptor	**Not directly modifiable**	HBufC8	const char* (pointing to owned data)

Descriptor memory layout (for the curious:-)



Basic usage (and please note the naming convention)

```
// Create a literal descriptor
_LIT(KTxtHelloWorld, "Hello World!"); //TLitC
const TInt KHelloWorldLength = 12;
// create a TBuf
TBuf<KHelloWorldLength> tbuf(KTxtHelloWorld);
// create a TBufC
TBufC<KHelloWorldLength> tbufc(KTxtHelloWorld);
// create a HBufC
HBufC* hbufc = KTxtHelloWorld().AllocLC();
// Create an RBuf
RBuf rbuf;
rbuf.CreateL(KHelloWorldLength, KTxtHelloWorld);
rbuf.CleanupClosePushL();
// create a TPtrC
TPtrC tptrc(tbufc);
// create a TPtr
TPtr tptr = *hbufc;
```

About bounds checking

- You can’t get buffer overflows with descriptor APIs!!
- Append, At, [], Format and Copy will panic your thread before you overwrite any data!
- Makes you fix it really quickly as opposed silently tripping over :-)
- ASCII to Unicode conversion comes for free with Copy()

…and much much more

Symbian OS C++ exceptions handling

Native Symbian OS C++ programs use an exceptions handling mechanism which is not compatible with the ISO C++ 0x exceptions handling way (actually the design of Symbian OS predates ISO/IEC 14882 1998/2003).

In Symbian OS, programmers have to think and implement their trapping, throwing and heap allocated object tracking in the face of exceptions, manually - as part of their design and not as an afterthought. (Test Error Driven Development :-)

Symbian OS C++ exceptions handling philosophy, is to be ‘’in your face’’, but not ‘’in the way’’.

Prehistory of Epoc exceptions

- Psion software was using exceptions circa ’89 with POOC
- Exceptions back then as with Symbian OS today are closer to how Objective-C works rather than how modern C++ works
- Complex Symbian OS C++ objects have to be on the heap and be manually tracked
- …this is because the cleanup mechanism doesn’t cater for stack allocated objects

About exceptions handling…

```
void SomeMethodL()
    {
    tmpObject = new (ELeave) CHeapyObject()
    CleanupStack::PushL(tmpObject)
    DoSomethingThatMayLeaveL()
    CleanupStack::Pop(tmpObject);
    // …and pass ownership of that object
    //OR possibly
    CleanupStack::PopAndDestroy(tmpObject)
    }
```

Trapping and Leaving

```
Cleanupstack::PushL(something);/* something will
   be tracked by the CleanupStack until poped */

TRAPD(err,SomeMethodL());

if (KErrNone!=err)
   {
    //start handling the exception
    //and possibly propagate it by another Leave
   }

…possibly pop something
```

How does the CleanupStack work ?

- It stores pointers to objects to be destroyed in case of an exception (a.k.a. Leave)
- These pointers are stored in nested levels
- Such levels are marked by the TRAP macro (think _almost_ of ‘C’ setjump/longjump here with exec call)
- When a Leave occurs, it makes sure it calls all d’tors (and cleanup items) of objects belonging to the corresponding Trap level.
- And the stack unwinds to the point of TRAP, returning an exception error code.

And why is that ?

- Poor or no support for C++ exception handling in compilers at the time
- CleanupStack is in your face → a good thing
- Compilers behind the scenes write all the exception handling code for you, they tend to be really conservative and churn much more code than a developer would …(challenge me on this, go on!:-)
- ….this may change with newer compilers…

(Almost) Introducing Modern ISO Standard C++…

(yielding to peer pressure that is)

- TRAP and User::Leave() are now implemented internally in terms of catch and throw
- Ported 3rd party code can use standard C++ exception mechanisms

  → try/catch/throw

  … But not mix these with Symbian OS system APIs

  … "Leaving functions must not throw, unless they also catch internally" → "barriers''
- Standard C++ exception specifications are supported

..more

- The C++ spec ISO/IEC 14882:1998/2003 is enabled but not delivered.
- RTTI and dynamic_cast<> is enabled, but not for Symbian OS APIs.
- RTTI - dynamic_cast<> implies the use of catch() to handle the exception that is thrown when the cast fails, and this doesn't mesh well with the limited use of C++ exceptions for TRAP and Leave().
- Current coding standards are to use NONSHARABLE_CLASS for internal component classes, which disables the RTTI info being emitted.

…therefore remember

- Symbian OS is now compiled with exceptions turned on, i.e. exception tables and the like are included in the (x86 and EABI) binaries
- implemented User::Leave/TRAP in terms of exceptions
- It is possible to throw and catch exceptions within your own code.
- It is possible to catch leaves and exceptions in your own code….Don't
- You cannot, in general, TRAP an exception
- Code that leaves and code that throws should not intermingle
- nor should code that leaves have objects on the stack with non-trivial destructors
- nor should code that throws, use the cleanup stack OR depend on code that indirectly depends on objects on it.
- Code that leaves MAY NOT call code that throws, without some suitable **barrier** -> will not call the cleanup sequence

On Barriers

- Q: Why do we need a 'suitable' barrier?
- A: To protect the integrity of the cleanup stack.

```
#ifdef __LEAVE_EQUALS_THROW__
EXPORT_C void User::Leave(TInt aReason)
   {
   TTrapHandler* pH = Exec::TrapHandler();
   if (pH)
        pH->Leave(aReason);// causes things on the cleanup stack to be
   cleaned up
   throw XLeaveException(aReason);
   }
#endif
```

A C++ teaser…. be prepared !

- Can you spot any problems ?

```
void SymbianFunc1L()
    {
    Cx* obj1 = Cx::NewLC();
    NonSymbianFuncL(obj1);
    CleanupStack::PopAndDestroy(obj1);
    }
void NonSymbianFuncL(Cx* aCx)
    {
    Sx obj2(aCx); // automatic object with destructor
    SymbianFunc2L(obj2.DoSomething());
    obj2.DoSomethingElse();
    // obj2 destroyed here
    }
void SymbianFunc2L( aParam )
    {User::Leave( leavecode );}
```

A small contribution to the C++ community

- Because of the efforts at Symbian to allow developers to ‘intermix’ C++ binaries, on Symbian OS v9.x, created with different compilers, ARM and Symbian standardised the new C++ EABI for all ARM processors.
- These efforts have been contributed to GCC-E which is now compatible with ARM’s RVCT.

Yes, Symbian OS was the guinea pig to verify this experiment, while we were building Symbian OS v9.0!

Thank you !

Q & A

For more:

- “Symbian OS Explained”, Jo Stichbury
- “Symbian OS for Mobile Phones vol2” , Richard Harrison
- “Symbian OS Internals: Real Time Kernel Programming”, Jane Sales et al
- “The Symbian OS Architecture Sourcebook: Design and Evolution of a mobile Phone OS”, Ben Morris
- “Symbian for Software Leaders: Principles of Successful Smartphone Development Projects2”, David Wood