Speedy and simple mobile music application development

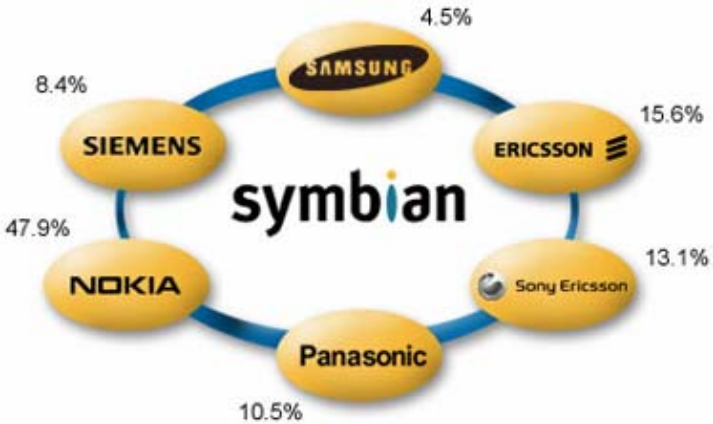
Part I—Introduction to Symbian/S60 audio programming
Part II—Writing real-time mobile apps with Python and C++
Part III—Mobile optimizations, case beat tracker

Jarno Seppänen
Nokia Research
2007-11-01

NOKIA
Connecting People

Symbian and S60

Symbian owners



- Symbian is a mobile operating system
  - 72% market share (of smartphones)
  - Other licensees: LG Electronics, Motorola, …
- S60 is a smartphone platform on top of Symbian
  - UI framework, apps (browser), runtimes (Python), API's (web services), …
  - 53% market share (of smartphones)
  - Licensees: LG Electronics, Lenovo, Nokia, Samsung 3rd

N95

- S60 3rd Ed. FP1
  - Symbian 9.2
- ARM11 @ 332 MHz
- 240x320 screen (QVGA)
- WLAN, 3G, GPS
- AAC+, AAC, MP3, WMA

N81

- S60 3rd Ed. FP1
  - Symbian 9.2
- ARM11 @ 369 MHz
- 240x320 screen (QVGA)
- WLAN, 3G
- AAC+, AAC, MP3, WMA

5500

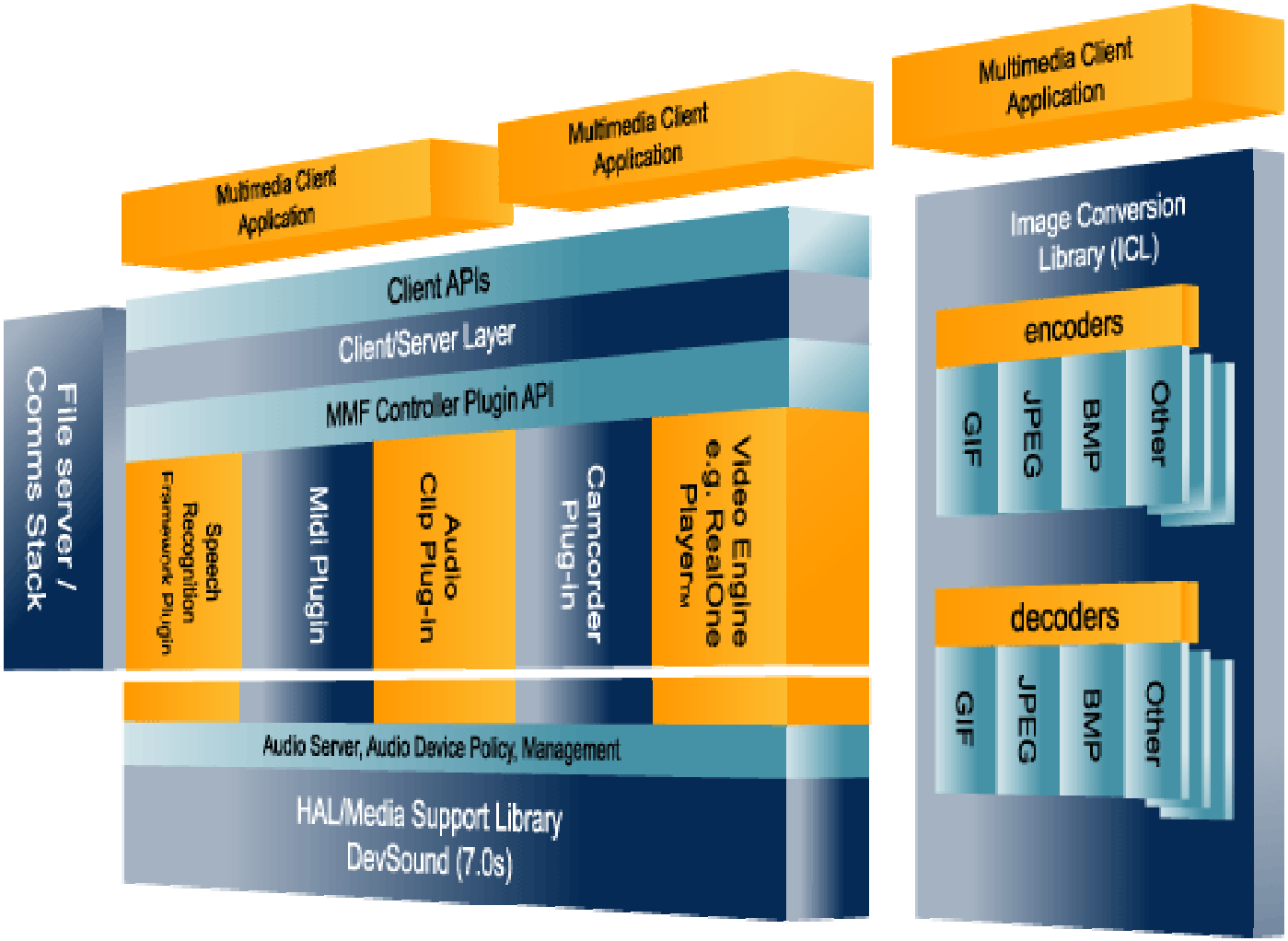
- S60 3rd Ed
  - Symbian 9.1
- ARM9 @ 235 MHz
- 208x208 screen
- GSM
- AAC+, AAC, MP3

NOKIA Connecting People

Symbian for multimedia apps

- Symbian devices have rich multimedia capabilities
  - *”Nokia is the biggest camera manufacturer and the biggest MP3 player manufacturer in the world…”*
  - Most smartphones have cameras and music players onboard
  - Next step: *world’s biggest GPS navigator manufacturer :-)*
- Increasingly more powerful hardware
  - Dual-chip systems, generic ARM CPU and dedicated DSP
  - CPU speeds and memory sizes are increasing
- Excellent connectivity
  - Many devices equipped with WLAN and 3G radios
  - Standards compliant web browser

Symbian multimedia architecture



NOKIA Connecting People

Symbian high-level audio API summary

- **CMdaAudioPlayerUtility**
  - Play audio files
  - Uses DSP decoders when available
- **CMdaAudioRecordingUtility**
  - Record audio files
  - Uses DSP decoders when available
- **CMdaAudioOutputStream**
  - Buffered audio stream playback
- **CMdaAudioInputStream**
  - Buffered audio stream capture
- Mobile specifics
  - Audio may be interrupted (e.g. incoming call)
  - Automatic mixing and volume control (e.g. received SMS notification)

Symbian high-level API summary continued

- Fancy audio features
  - Audio effects and enhancements: **CAudioEqualizer**, **CBassBoost**, **CEnvironmentalReverb**, ...
  - Speech recognition: **CSpeechRecognitionUtility**
  - Text-to-speech: **CMdaAudioPlayerUtility**
  - MIDI client: **CMidiClientUtility**
  - DRM music player: **CDrmPlayerUtility**
- Other multimedia related
  - Optical character recognition: **MOCREngineInterface**
  - Camera: **CCamera**
  - GPS location: **RPositioner**
  - OpenGL ES

Documentation pointers

- Symbian developer network
  - Symbian API docs
    - http://developer.symbian.com/main/oslibrary/osdocs/
- Forum Nokia
  - S60 API docs
    - http://www.forum.nokia.com/document/Cpp_Developers_Library/
  - Whitepapers
    - Multimedia Framework Architecture in S60 Devices (2007)
    - S60 Platform: Music Application Developer's Guide (2006)
    - Symbian OS: Multimedia Framework And Other Multimedia APIs (2005)
    - Symbian OS: Creating Audio Applications In C++ (2005)
    - S60 2nd Edition Feature Pack 2: OpenGL ES 3D Example (2005)

Speedy and simple mobile music application development

Part I—Introduction to Symbian/S60 audio programming
Part II—Writing real-time mobile apps with Python and C++
Part III—Mobile optimizations, case beat tracker

Jarno Seppänen
Nokia Research
2007-11-01

Motivation

- Music applications have two complementary halves
  - Score processing/signal processing: both are essential but opposite
  - Asynchronous/ synchronous, event-driven/stream-driven, dynamic/static
  - Cf. Csound score and orchestra, MIDI and DLS, ...
- Python and C++ are a win–win for rapid development of real-time mobile apps
  - Generic solution, not only for music
  - Easy-ish prototyping, as long as no dedicated mobile music frameworks available
- Python is a dynamic high-level language
  - Fast-writing: clear, simple, and fun; dynamic typing; automatic memory mgmt
  - Slow-running: inappropriate for real-time numerical computation
- C++ is a static high-level language
  - Slow-writing: verbose and complicated (especially Symbian API's)
  - Fast-running: good compiler support; static typing; manual memory mgmt

Dive in

Users dive in

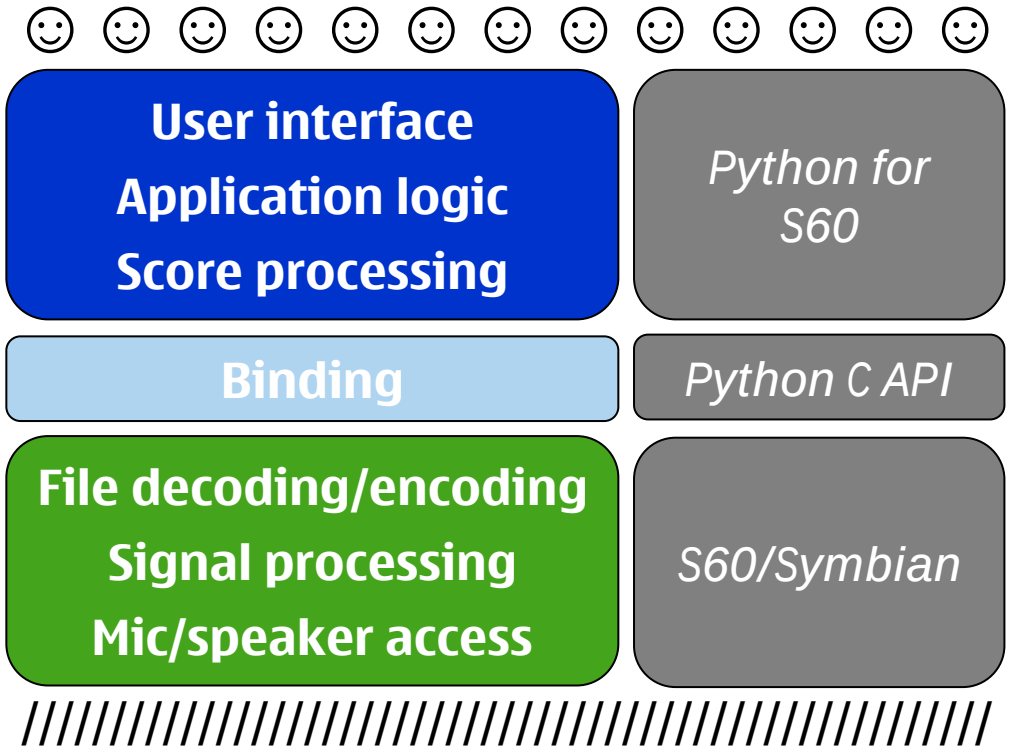
1. install **Python for S60** and **Python Script Shell** to phone
2. install **PocketSaw** to phone
3. launch Python Shell, run **PocketSaw.py**

Hackers dive in

1. install S60 & PyS60 SDK's
2. build WINSCW
3. run in emulator
4. build ARMV5
5. package & sign
6. install to phone
7. launch

Two-layer architecture

- Top layer
  - UI, application logic
  - Score processing (dynamic, event-driven)
  - Python
- Bottom layer
  - Real-time computation
  - Signal processing (static, stream-driven)
  - C++
- Layer binding
  - The interface between Python and C++
- Top/bottom partition
  - Design binding, choose abstraction level
  - **Try to minimize sizes of both the binding and the bottom half**
  - Careful thought required



Python for S60

- Python interpreter ported for S60 devices
- Bindings to several S60 API's
- History
  - 1.0 released in 2004
  - 1.2 released Oct 2005
  - 1.4 released Jul 2007
- Status
  - Mature, quite widely used
  - Several third-party extension modules available
- Comes in two parts
  - Python for S60 runtime
  - Python Script Shell application

Python for S60 GUI basics

- Really simple GUI creation
  - import appuifw, e32
    lock=e32.Ao_lock()
    def select(): print 'click!'
    appuifw.app.body=appuifw.Listbox([u'Apples', u'Oranges'], select)
    appuifw.app.exit_key_handler=lock.signal
    lock.wait()
- In my experience, Listbox is all you need :)
  - Its **bind** method allows catching key presses
- Other useful GUI elements
  - **appuifw.app.menu**: list of menu items
  - **appuifw.app.title**: title text
  - **appuifw.note**, **appuifw.query**: dialogs
  - **appuifw.InfoPopup**: pop-up tooltips (new!)
    - **show**(text: unicode)

Python for S60 API features

- Rich S60 API support built in
  - **appuifw**: S60 user interface
  - **audio**: playback, recording, TTS
  - **sensor**: accelerometer
  - **location**, **positioning**: GPS, GSM cell info
  - **graphics**: 2D images
  - **glcanvas**, **gles**: 3D graphics
  - **inbox**, **messaging**: SMS, MMS
  - **camera**: photos and video
  - **telephone**: call handling
  - **calendar**: calendar access
  - **contacts**: phonebook access
  - **httplib**, **math**, **thread**, **socket**, **zlib**, ...: standard Python libs

Symbian/S60 C++ development

- Symbian C++ is a language (dialect) of its own
  - Own object system ("two-phase constructor", "CBase inheritance")
  - Own memory handling ("cleanup stack")
  - Own exceptions ("leaves", "traps")
  - Own string types (HBuf, HBufC, TPtr, TPtrC, TDes, TDesC, plus 8-bit and 16-bit variants...)
  - **No writeable static data**
  - **Incomplete C standard library (libc), missing C++ std library (STL) implementations**
- Good news: most of the above are just idioms
  - You can safely ignore when writing own code
- Minimal development setup
  - S60 C++ platform SDK
  - Python for S60 SDK
  - Command line build, "WINS" PC emulator
- Different compiler flavors
  - GCCE: Free GCC compiler
  - ARMV5: Commercial ARM RVCT compiler
- IDE recommended
  - CodeWarrior, Carbide C++
  - Even on-device debugging supported
- Choose S60 SDK according to your target devices
  - 2nd edition
  - 3rd edition
  - 3rd edition, feature pack 1

NOKIA Connecting People

PocketSaw

- Minimal real-time synthesis example
- Let's walk thru the details
- Start from http://musicdsp.org/files/synthesis002.txt
  - *"Bandlimited synthesis of sawtooth by leaky integration of a DSF BLIT"*
  - Implementation in floating point C code
- Original *synthesis002* code has a two-method interface
  - **init_blit(blit_t, anq, cutoff):** initialize the blit_t data structure with given parameters
  - **update_blit(blit_t, ω) → double:** compute an audio sample, given normalized fundamental frequency ω

Python/C++ interface

- Choose Python/C++ partition and design binding interface
- Create Python **Saw** class:
  - **Open(anq, cutoff):** start audio playback (mono 48 kHz)
  - **Close():** finish audio playback
  - **SetVolume(vol):** change audio volume
  - **SetFrequency(freq):** change fundamental frequency in Hertz
- Actual C++ **Saw** implementation:
  - Implement above interface methods
  - Implement Symbian audio driver
  - Call *synthesis002* **init_blit** and **update_blit** methods for signal processing

Python/C++ binding

- Write **saw** Python extension module
  - Source code **saw.cpp**
  - Binary **saw.pyd** (a renamed DLL)
  - Contains the **Saw** class
- Usage
  - import saw —*load C++ module*
    s=saw.Saw() —*construct Saw synthesizer*
    s.Open(0.5, 0.0001) —*choose parameters and start audio*
    s.SetVolume(0.5) —*halfway between silence and max volume*
    s.SetFrequency(440) —*fundamental frequency in Hertz*
    sleep(10) —*wait 10 seconds (with audio playing)*
    s.Close() —*stop audio*
- Fixed sample rate (48 kHz), channels (mono), and frame length (1000 samples)

User interface

- Write PocketSaw.py
- View instrument options as a Listbox
- Bind c-major scale to keypad buttons 1–9
- Bind volume controls to keypad * and # buttons

The boring part

- Write Symbian audio driver
  - Implement **MMdaAudioOutputStreamCallback** methods for audio playback
  - Call **CMdaAudioOutputStream** for audio set-up
- Write Python wrapper
  - Convert between Python and C++ types
  - Convert between C++ errors and Python exceptions
  - Manage object reference counting
- Set up Symbian build and packaging
- D'oh: remove writeable static data
  - When porting standard C code, may potentially mean substantial code rewrite
  - Here, change **double pi** to **const double pi**

NOKIA Connecting People

Detailed contents

- Files
  - BWINS/saw.pyd.def —*Symbian build*
    EABI/saw.pyd.def —*Symbian build*
    PocketSaw.pkg —*Symbian packaging configuration*
    grp/bld.inf —*Symbian build*
    grp/saw.mmp —*Symbian build makefile*
    js.cer —*Symbian packaging certificate*
    js.key —*Symbian packaging certificate*
    pocketsaw.py —*Python application in ~100 lines*
    saw.cpp —*C++ engine in ~300 lines*
    synthesis002.c —*C bandlimited sawtooth algorithm in ~150 lines*
- Building, packaging, and signing
  - bldmake bldfiles —*Set-up*
    abld build armv5 urel —*ARMV5 release build*
    abld build winscw udeb —*WINSCW debug build*
    makesis PocketSaw.pkg PocketSaw_unsigned.sis —*Create installer*
    signsis PocketSaw_unsigned.sis PocketSaw.sis —*Sign installer*

NOKIA Connecting People

PocketSaw open issues

- Audio clicks: sawtooth algorithm is is too heavy for ARM
- Implemented in floating point
  - Rewrite in fixed point left as an exercise :)
- Using sin, cos, and pow functions in audio computation
  - Replace with lookup tables
- Using division in audio computation
  - Precompute reciprocals where possible

Python extensions and C API

- Standard C API for creating Python extension modules
  - Creating modules, classes: **Py_InitModule**, **PyModule_AddObject**, **PyObject_New**
  - Modifying python objects: **PyArg_***, **PyTuple_***, **PyList_***, **PyInt_***, **PyFloat_***
  - Calling methods: **PyEval_***
  - Reference counting: **Py_INCREF**, **Py_DECREF**
  - Exception handling: **PyErr_***
- Also used to create C++ extensions for Python on S60
  - Symbian specific solution for global/static data: **SPyGetGlobalString**, **SPyAddGlobalString**
- Documentation
  - *Extending Python with C or C++*
  - http://www.python.org/doc/ext/intro.html

NOKIA Connecting People

Python for S60 advanced topics

- Standalone installation package creation
  - Python apps looking like native apps: install from single SIS file; add own icon in Applications menu
  - Tools available: py2sis and Ensymble
- Callbacks and the Global interpreter lock (GIL)
  - All python code runs with GIL locked
  - C++ modules may release GIL during blocking system calls (e.g. network comms)
  - Callbacks from C++ code must acquire GIL before using Python C API
- Active objects
  - Symbian API favors co-operative multitasking instead of threads
  - Several Active Objects run inside one thread
- Security certificates
  - When using DLL's, they need equal or greater capabilities than the caller
  - For example, python pyd's require equal capabilities to the main python DLL

Speedy and simple mobile music application development

Part I—Introduction to Symbian/S60 audio programming
Part II—Writing real-time mobile apps with Python and C++
Part III—Mobile optimizations, case beat tracker

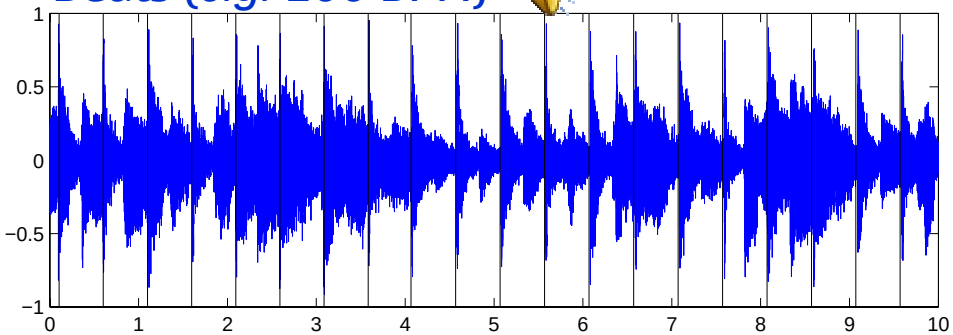
Jarno Seppänen
Nokia Research
2007-11-01

NOKIA
Connecting People

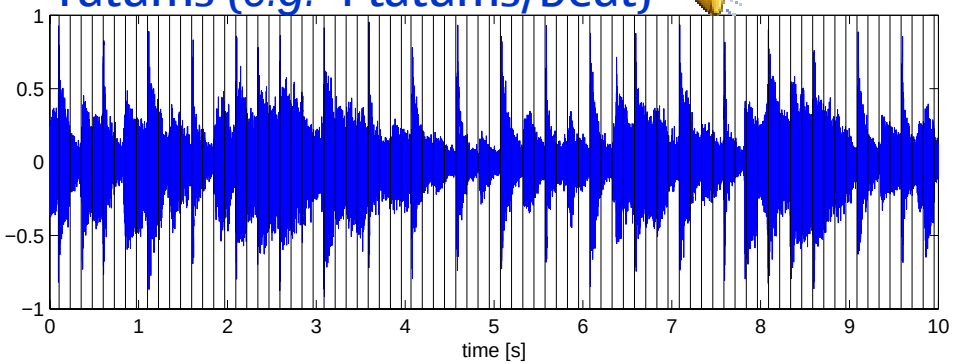
Algorithm overview

- Extract pulses on two key metrical levels: **beat** and **tatum**
  - Sequences of beat and tatum times
- Causal mode of operation
  - ~4 second analysis latency
- Algorithm design targets
  1. **Target human beat tracking results** on selected material
  2. **Focus on mainstream music**, music with a strong and steady beat
  3. **Improve computational efficiency** to enable mobile implementations

Beats (*e.g.* 100 BPM)



Tatums (*e.g.* 4 tatums/beat)



NOKIA Connecting People

Computation stages

- The beat & tatum tracker runs in six stages
  1. resampling stage
  2. accent filter bank stage
  3. buffering stage
  4. periodicity estimation stage
  5. beat and tatum period estimation stage
  6. beat phase estimation stage

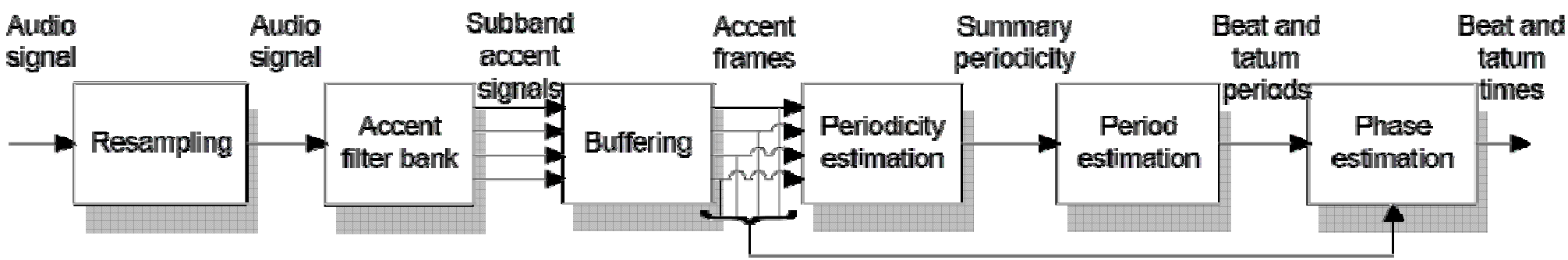


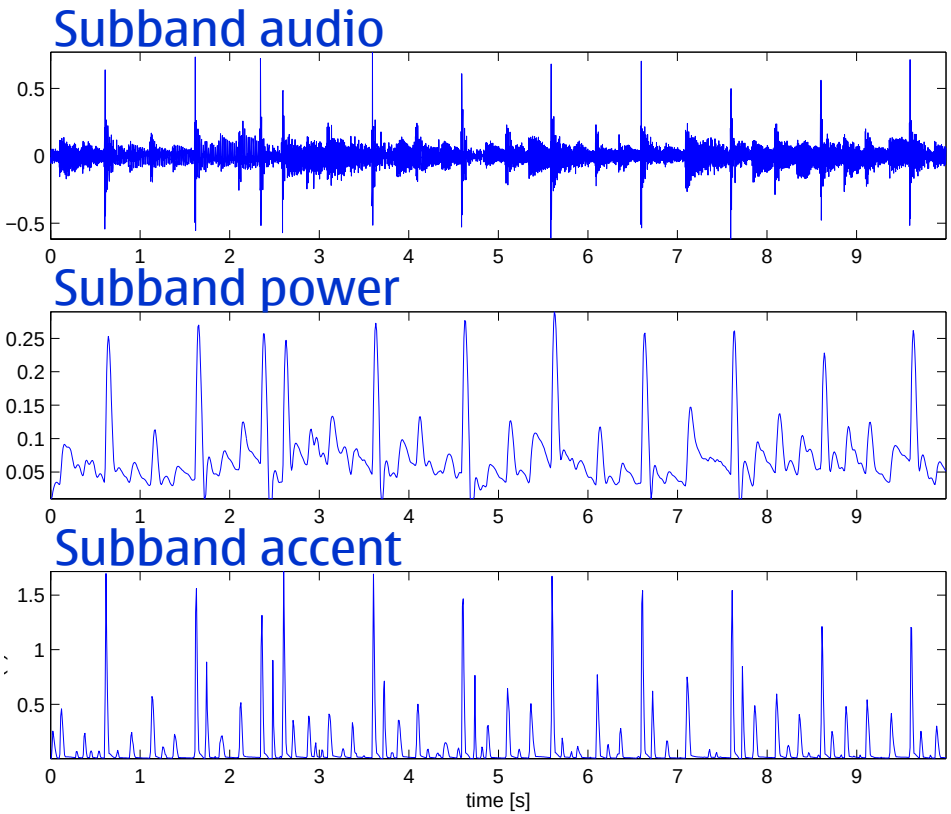
Figure 1. Beat and tatum analyzer.

Resampling stage

- All incoming signals are resampled
- Accent filter bank designed for 24 kHz fixed sample rate
- Uses a coarse algorithm: first-order interpolation
  - Audio fidelity doesn't matter much
  - Artifacts are OK as long as the **metrical percept** doesn't change
  - Slight degradation of precision

Accent filter bank stage

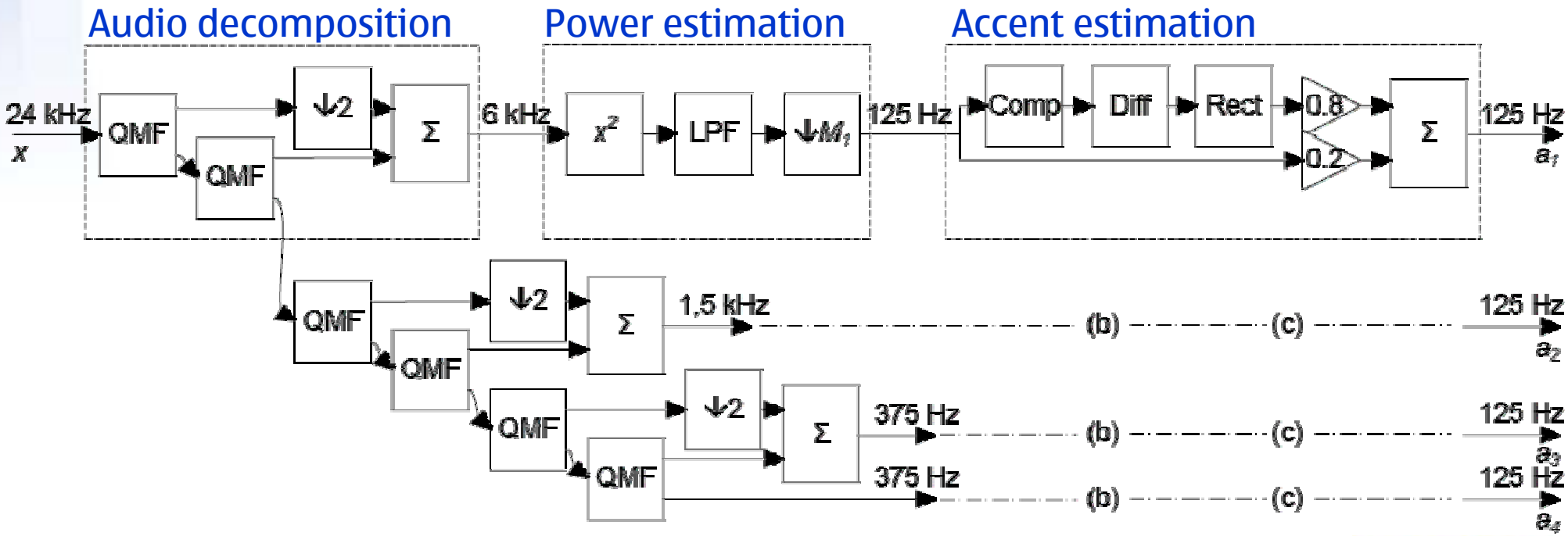
- Signal preprocessing for rhythm information
- Extract accent information
  - *Phenomenal accents:* musical stress, loudness
  - E.g. sound onsets
- Discard other information as irrelevant
  - Harmonic structure
  - Timbre
  - Melody
- Most critical stage efficiency-wise
  - Most raw audio processing of all six stages
- Signal data reduced 48-fold
  - 24 kHz audio input, four 125 Hz accent outputs



NOKIA Connecting People

Accent filter bank details

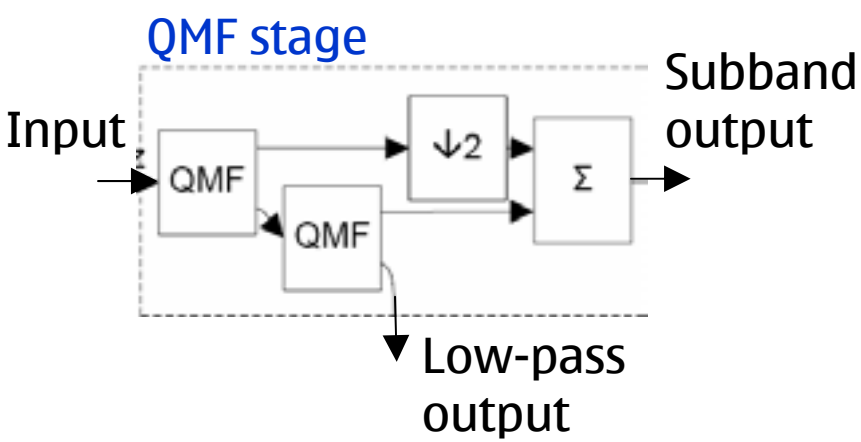
- Audio decomposed into frequency bands
  - Four bands
  - Two octaves each
- Power estimation
  - 10 Hz low-pass filter
  - Downsampling by *M*
- Accent estimation
  - Nonlinear compression (Comp)
  - First-order differentiation (Diff)
  - Half-wave rectification (Rect)

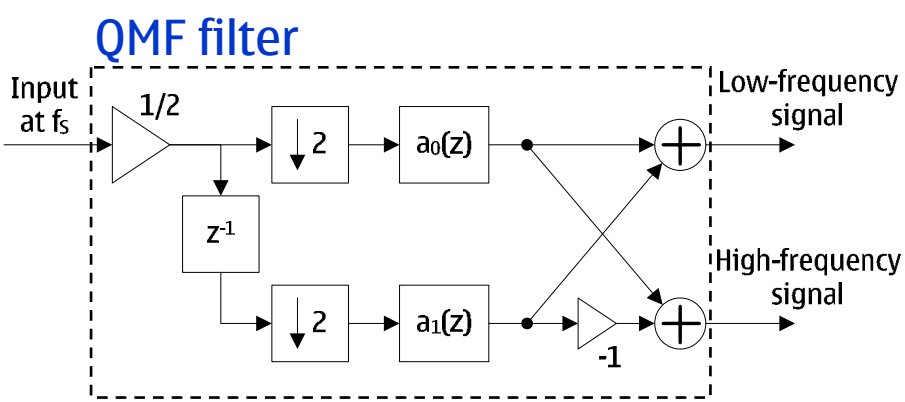


NOKIA Connecting People

QMF filtering

- Six QMF filters cascaded in three QMF stages
  - Seven intermediate frequency bands
  - Quadrature mirror filters
- First-order prototype filters $a_0(z)$ and $a_1(z)$ make the structure computationally very efficient
  - Sufficient signal-to-noise ratio for music meter estimation purposes
  - **Not a high-fidelity application! ☺**
- Downsampling by two creates aliasing noise, but that's OK as well





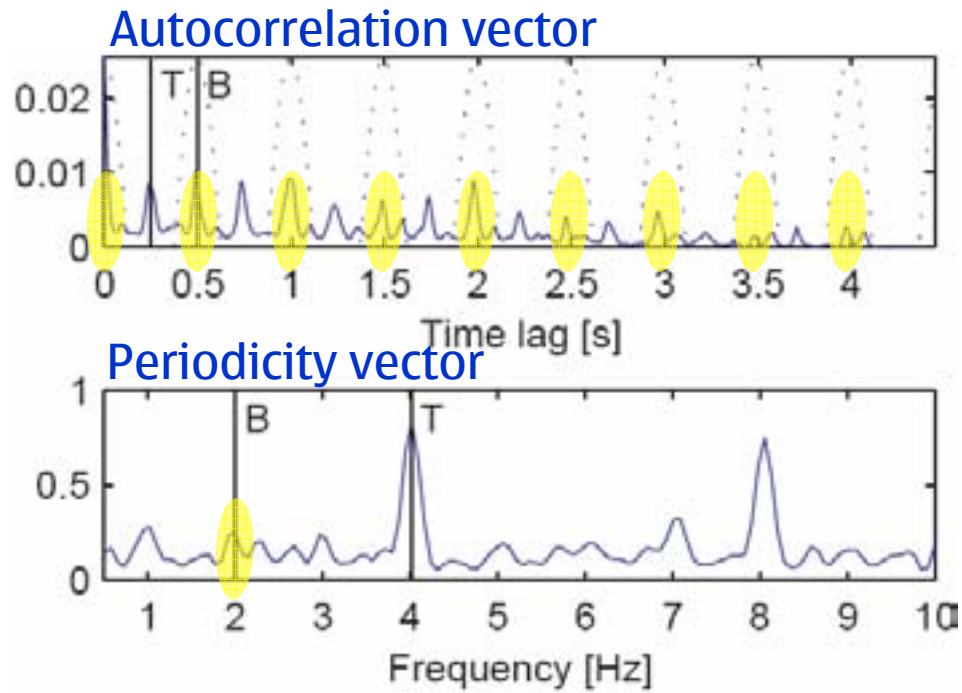
Buffering stage

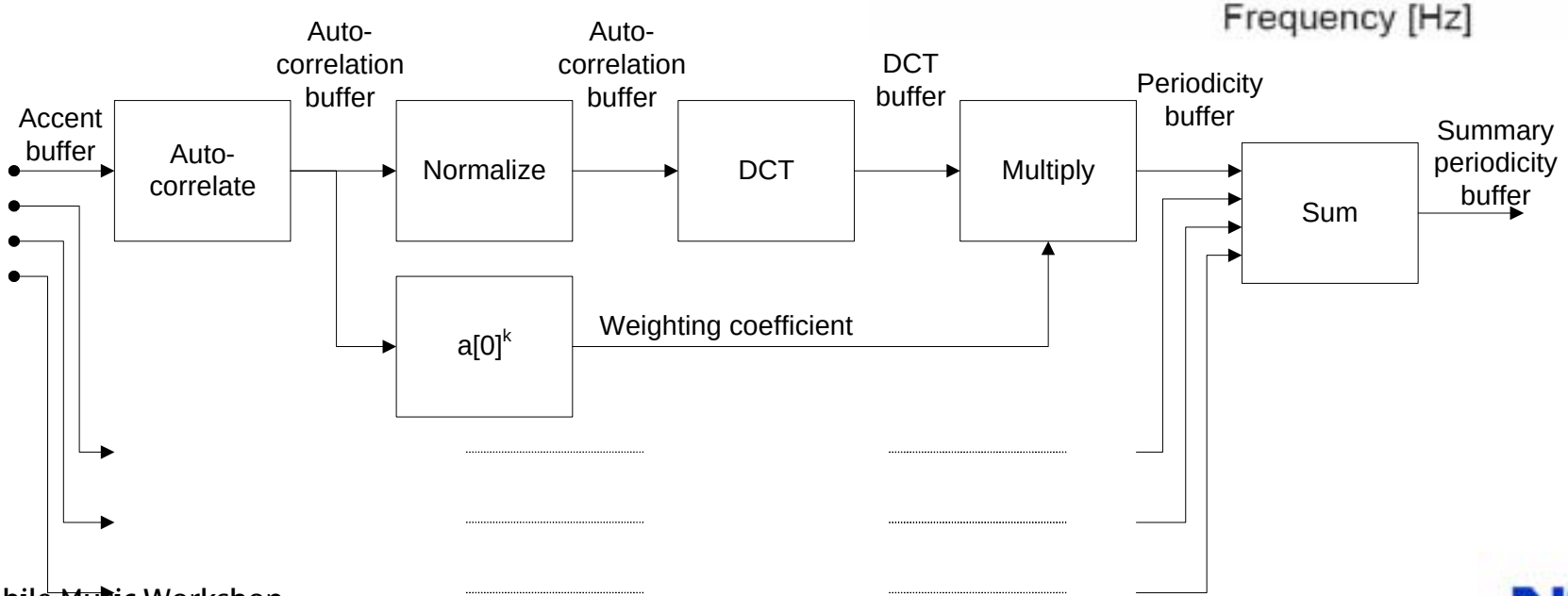
- Buffering stage converts incoming audio frames to fixed frame length
- Back end frame length 512 samples
  - 4,1 seconds at 125 Hz accent signal rate
- Buffer handover between front end and back end

NOKIA Connecting People

Periodicity estimation stage

- Analyze accent signals for repetition
- Transform subband accent into *periodicity*
  - Autocorrelation combined with DCT (discrete cosine transform)
  - Comparable to IOI (inter-onset interval) histogramming methods
- Summary periodicity across bands

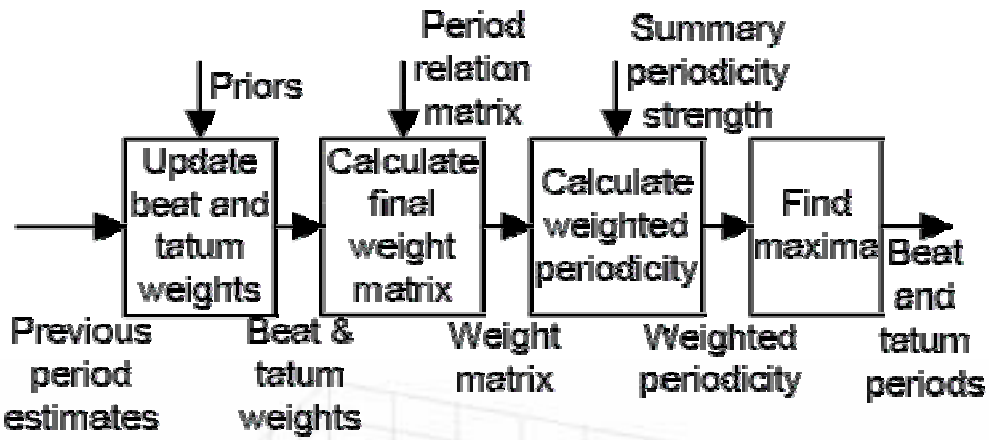


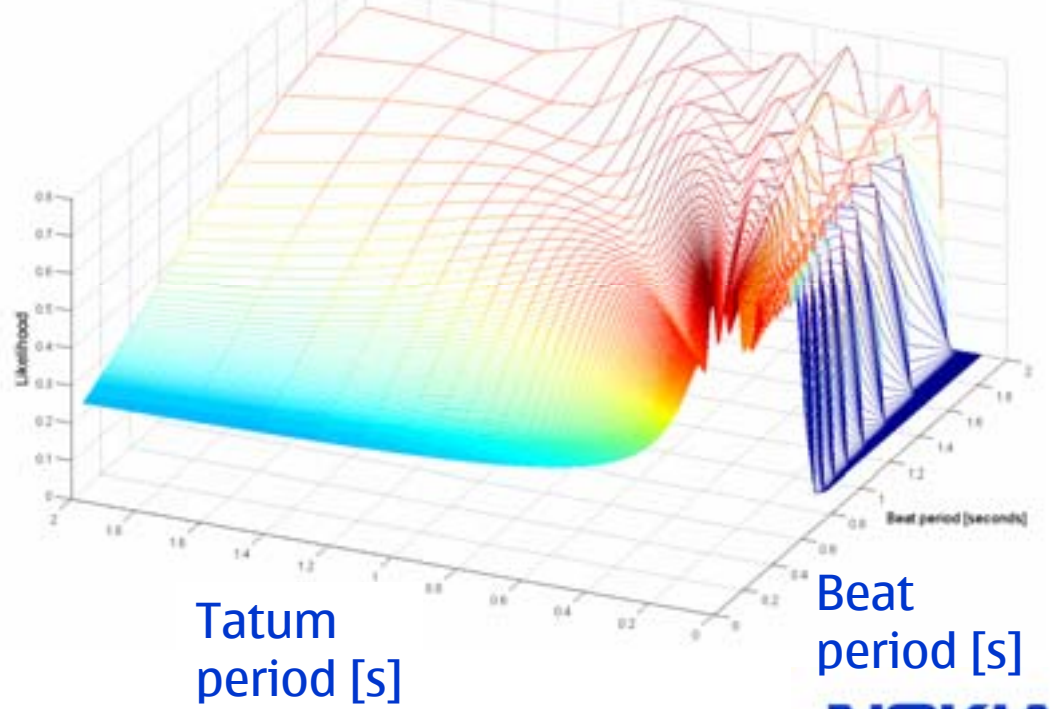


NOKIA Connecting People

Beat and tatum period estimation stage

- Beat and tatum periods chosen based on summary periodicity
  - Once every 4,1 seconds
- **A priori period distributions**
  - Log-normal distribution shape
  - Experimental parameter values, see Parncutt (1994)
- **Inter-frame period continuity**
  - Log-normal distributions
  - Experimental parameters, see Klapuri *et al.* (2006)
- **A priori beat–tatum period relations**
  - Mixture of Gaussians
  - Experimental parameters, see Klapuri *et al.* (2006)

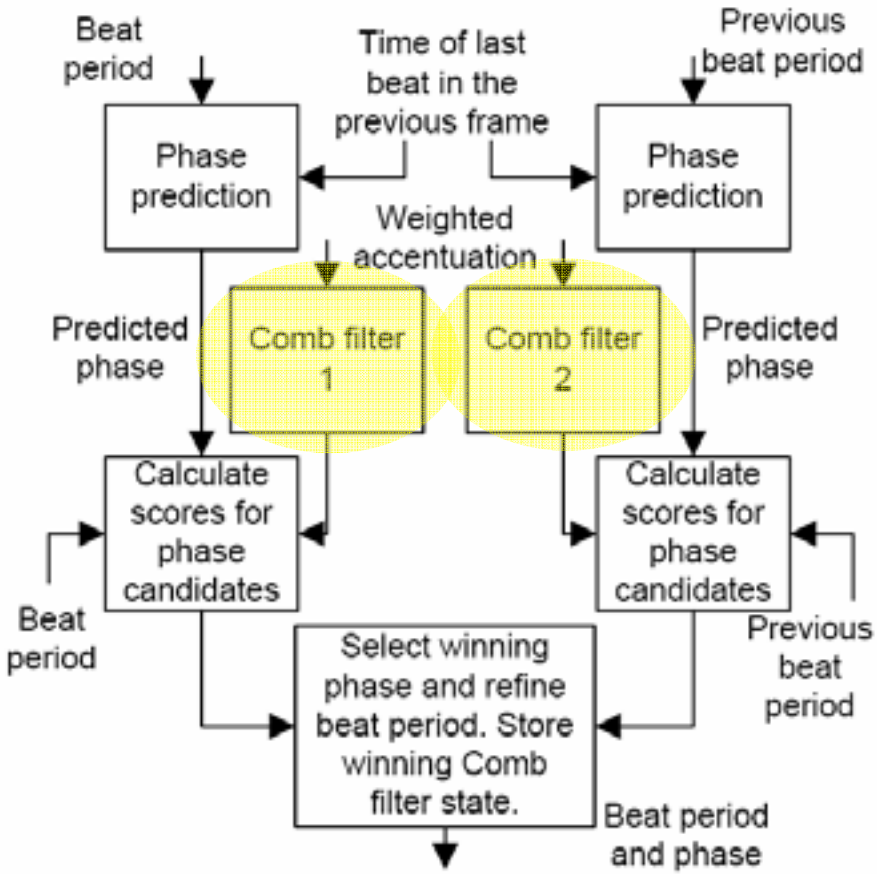




NOKIA Connecting People

Beat phase estimation stage

- Pulsation followed by adaptive comb filtering
  - Comb filter delay controlled by period
  - Comb filter response acts as phase goodness measure
- Only estimate beat phase
  - Tatum phase is same as beat phase
  - Metrical hierarchy
- Use two comb filters for two phase/period hypotheses
  - Current phase and previous phase
  - Using current period and previous period
  - Cf. earlier approaches (bank of filters)



Beat tracker evaluation

- Beat tracking accuracy
  - Beat period & phase agreement with ground truth
- Processing efficiency
  - Median of five runs for 30 second clip
  - CPU cycles for 1.86 GHz Pentium M
- 192 one-minute song excerpts
  - Mainstream music with steady beat
  - Ground truth: manual beat annotation

- Klapuri *et al.* (2006)
  - Beat, tatum, and bar estimator (three metrical levels)
  - Original author's implementation
  - Mixed MATLAB/C++ implementation (61% MATLAB, 39% C++ cycles)
- Scheirer (1998)
  - Beat tracker
  - Original author's implementation
  - Floating-point C++ code

Table 1. Beat tracking accuracy scores.

	Continuity required			Individual estimates		
Method	Correct	Accept d/h	Period	Correct	Accept d/h	Period
Proposed	60%	70%	76%	64%	76%	79%
Klapuri	66%	76%	73%	72%	85%	81%
Scheirer	29%	34%	30%	53%	65%	59%

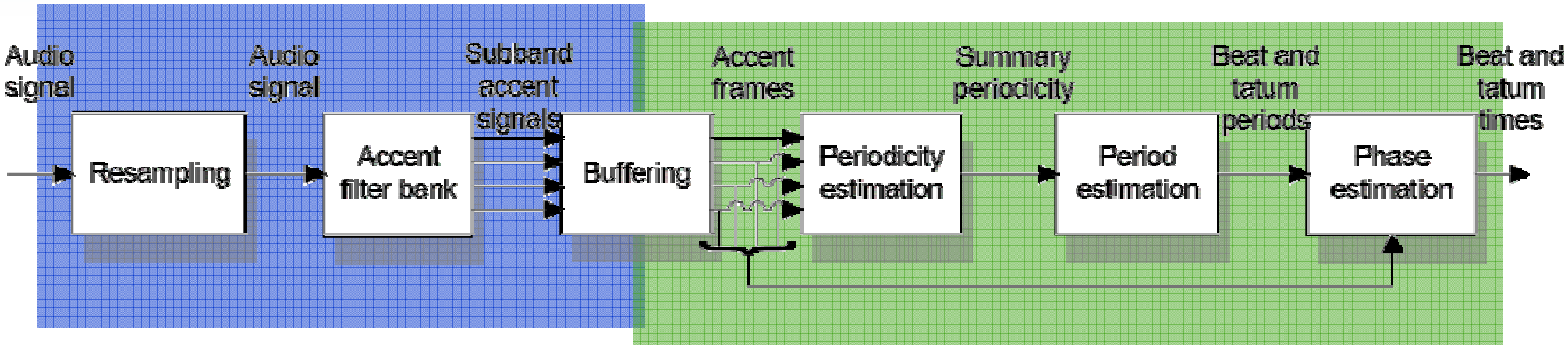
Table 2. Processor usage profiles.

Method	Mcycles	Mcycles/s
Proposed	678	2.3 !!!
Klapuri (MATLAB)	125000	420
Scheirer	136000	450
Scheirer without `malloc` etc.	119000	390

NOKIA Connecting People

Implementation front end–back end partitioning

- The algorithm is partitioned into two frame lengths
- **Front end** runs synchronously with incoming audio
  - Freely changeable frame length
  - Accumulate data into fixed frames
- **Back end** runs asynchronously
  - Process frames as they arrive from front end
  - Fixed frame length of 4 seconds



Mobile demonstration: dancer

Audio implementation rules

- Fixed point is your friend
  - Necessary for real apps
- Use look-up tables
  - All trig functions, exponentials, logs, etc.
- Use correct data types (integers, 16 vs 32 bits)
- Eliminate function calls
- Unroll loops

Practical optimization cycle

1. Write C++
2. Compile to assembly
3. Read assembly, find inefficiencies
4. Optimize C++
5. Goto 2

- If all else fails, ARM9E and later have DSP intrinsics
  - **#include <armdsp.h>**
  - smulwb—16x32 bit multiply
  - smlawb—16x32 bit multiply and accumulate
  - qadd—add with saturation

- ARM chip documentation
  - Assembly summary: *ARM/Thumb Instruction Set Quick Reference Manual*
  - Details: *Technical Reference Manual*
- Relevant chips
  - 5500—ARM9E@235 MHz
  - N81—ARM11@369 MHz
  - N95—ARM11@332 MHz
  - More specs at http://www.forum.nokia.com

NOKIA Connecting People

Questions?

NOKIA
Connecting People