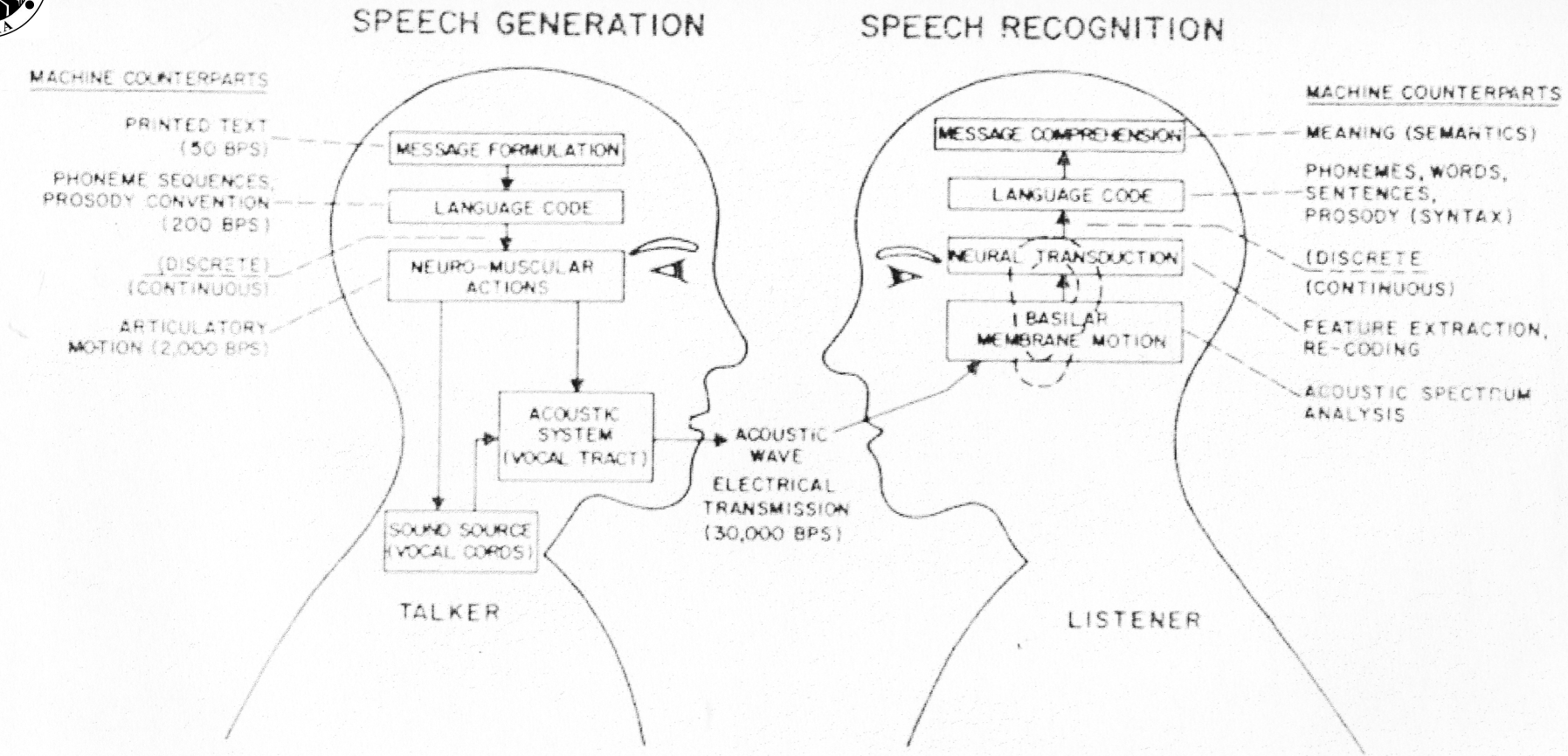
2. TEORI PENUNJANG

2.1. Proses Produksi dan Persepsi Percakapan pada Manusia

Gambar 2.1. memperlihatkan suatu diagram skematis dari proses produksi dan persepsi percakapan pada manusia. Proses produksi (proses untuk menghasilkan percakapan) dimulai ketika seseorang (si pembicara) memformulasi suatu pesan (di dalam pikirannya) yang ingin ia sampaikan kepada orang lain (si pendengar) melalui percakapan. Langkah selanjutnya pada proses ini adalah konversi pesan menjadi suatu kode bahasa. Sesaat setelah kode bahasa ditentukan, si pembicara harus melaksanakan suatu rangkaian perintah *neuromuscular* yang akan menyebabkan saluran-saluran vokal bergetar bila diperlukan dan membentuk bidang vokal sedemikian hingga deretan suara percakapan yang diinginkan dihasilkan dan diucapkan oleh si pembicara, menghasilkan suatu sinyal akustik sebagai hasil akhirnya. Perintah-perintah *neuromuscular* tersebut harus secara serentak mengatur semua aspek gerakan yang berhubungan dengan artikulasi, termasuk dalam hal ini mengatur bibir, rahang, lidah, dan *velum* (bagian belakang langit-langit di dalam mulut, disebut juga sebagai "*trapdoor*", yang mengatur aliran akustik yang berperan dalam mekanisme menghasilkan bunyi sengau).

Ketika sinyal percakapan telah dihasilkan dan diteruskan kepada si pendengar, maka dimulailah proses persepsi (proses untuk mengenali percakapan). Pertama-tama si pendengar akan memproses sinyal akustik di sepanjang membran *basilar* di bagian sebelah dalam dari telinga (bagian *inner ear*), yang berfungsi untuk menyediakan suatu analisa spektrum terhadap sinyal yang masuk. Selanjutnya, sebuah proses transduksi *neural* akan mengkonversi sinyal *spectral* yang diterima oleh bagian output membran *basilar* menjadi sinyal-sinyal aktifitas (*activity signals*) pada syaraf pendengaran. Kemudian aktifitas *neural* bersama-sama dengan syaraf pendengaran akan dikonversi menjadi suatu kode bahasa untuk diproses lebih lanjut pada bagian-bagian atas otak, dan akhirnya pemahaman terhadap pesan akan dihasilkan.



Gambar 2.1. Diagram Skematis Proses Produksi dan Persepsi Percakapan pada Manusia
Sumber: Pelton, G.E.(1996). Voice Processing. New York: McGraw-Hill.

2.2. Sejarah Perkembangan Sintesa Suara Manusia

Berbagai teknologi yang berhubungan dengan sintesa suara manusia yang telah banyak bermanfaat pada saat ini, pada awal perkembangannya tidaklah berasal dari satu akar, akan tetapi dari bebeberapa bidang ilmu. Dua bidang ilmu yang telah berkembang secara luas dan cepat yaitu bidang sains komputer (*computer science*), yang muncul pertama kali pada tahun 1940, dan bidang *telephony*, yang dimulai sekitar tahun 1876. Sintesa dan pemrosesan suara manusia tidak akan pernah berkembang bila tidak ada kontribusi dari mereka yang selama berabad-abad berusaha keras memahami mekanisme bagaimana manusia berbicara, dan mereka yang bereksperimen dengan berbagai alat dan mesin untuk menghasilkan suara dan percakapan manusia. Individu-individu yang memberikan kontribusinya dalam sejarah perkembangan bidang ilmu ini berasal dari berbagai profesi dan latar belakang yang bervariasi, yang sebagian di antaranya secara mengejutkan merupakan profesi atau golongan yang tidak terduga, yaitu antara lain pelaku akademis, golongan pendeta, artis atau seniman, mekanis, insinyur, ilmuwan, dan bahkan golongan klenik (*charlatans*).

Berikut ini akan dibahas secara singkat dua individu yang memiliki kontribusi penting terhadap sejarah perkembangan sintesa dan pemrosesan suara manusia.

2.2.1. Profesor Kratzenstein

Pada akhir abad ke delapan belas, ketertarikan untuk mempelajari mekanisme dari artikulasi tumbuh dimana-mana. Aktivitas ilmiah yang mendukung fonema tersebut adalah adanya beberapa usaha yang serius untuk membuat mesin buatan yang dapat berbicara. Pada tahun 1779, Imperial Academy of St. Petersburg mengajukan 2 (dua) buah pertanyaan untuk kompetisi tahunan yang diselenggarakannya:

1. “Apa yang menjadi karakter dan sifat dasar suara-suara yang dihasilkan oleh vokal a, e, i, o, u yang membedakannya dari yang lain?”
2. “Dapatkah suatu instrumen dirancang dan dibuat menjadi seperti pipa-pipa *vox humana* dari organ tubuh manusia, yang dapat secara akurat mengekspresikan suara-suara vokal?”

Seorang warga negara Rusia bernama Christian Gottlieb Kratzenstein, yang lahir di Jerman dan suatu ketika menjadi professor psikologi di Halle dan kemudian di Copenhagen, memenangkan hadiah dengan mengkonstruksi suatu rangkaian resonator akustik yang mensimulasikan mulut manusia. Resonator-resonator tersebut bervariasi bentuknya, bergantung kepada vokal yang akan ditirukan, dan semuanya saling terkait oleh suatu alat berupa buluh-buluh yang bergetar untuk menirukan paduan nada vokal yang dihasilkan manusia. Rangkaian tersebut, demikian dilaporkan, dapat menghasilkan setiap vokal dengan "tingkat akurasi yang dapat ditoleransi".

Robert Willis, ketika melakukan penelitian terhadap produksi vokal sintetis pada tahun 1829, menunjukkan bahwa Kratzenstein seharusnya dapat memperoleh hasil yang sama menggunakan sebuah tabung tunggal yang panjangnya disesuaikan untuk setiap vokal. Adalah suatu hal yang menarik bila tipe buluh yang ditemukan oleh Kratzenstein untuk kompetisi tersebut, kemudian hari digunakan pada harmonika. Pada tahun 1786 Kratzenstein menunjukkan kepada kenalan-kenalannya di Paris bagian dari mesin bicara yang telah dibuatnya menggunakan sebuah tuas penjepit silinder. Mesin tersebut hanya dapat menyuarakan vokal dan sedikit kata seperti "mamma" dan "papa".

2.2.2. Wolfgang von Kempelen

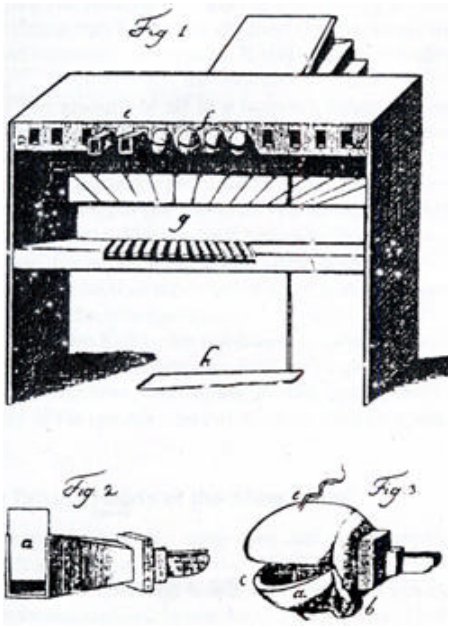
Selama periode 2 (dua) decade, dari 1769 sampai 1790, Wolfgang Ritter von Kempelen dari Wina, Austria menciptakan mesin bicara lengkap yang pertama. Dalam proses pembuatannya, ia membangun 3 (tiga) model yang berbeda, semuanya dioperasikan dengan tangan, dan pada tahun 1791 mendeskripsikan pekerjaan ini secara menyeluruh dan mendetail dalam sebuah buku.

Wolfgang von Kempelen berpendapat bahwa untuk membuat sebuah mesin bicara, ia harus terlebih dahulu dapat menghasilkan suara-suara vokal. Ia memulai dengan mencari sebuah sumber suara yang tepat, sebuah pengganti mekanis dari paduan nada suara-suara vokal. Ia mempertimbangkan buluh-buluh getar yang digunakan dalam instrumen-instrumen musik dan, meskipun tidak memuaskan secara keseluruhan, menempatkannya pada buluh berdengung yang

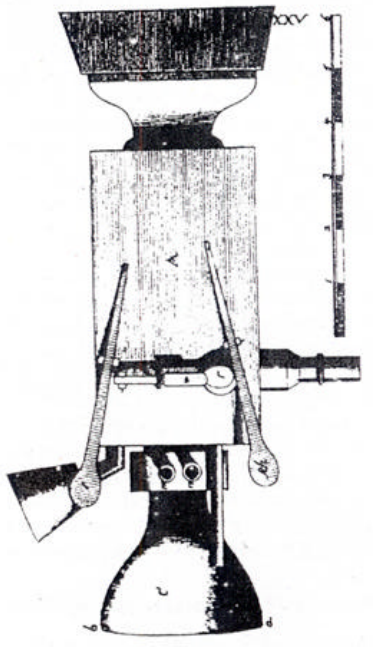
digunakan dalam kantung-kantung pipa sebagai satu-satunya cara yang menghasilkan suara yang paling menyerupai yang dihasilkan manusia. Suara dari buluh tersebut dimainkan ke dalam sebuah alat yang berbentuk lonceng dengan sebuah tabir pengatur pada bagian mulutnya, yang digerak-gerakkan untuk menghasilkan suara vokal yang berbeda-beda. Tidak puas dengan hasilnya, von Kempelen menggunakan telapak tangannya menggantikan tabir pengatur, meskipun lebih baik, suara-suara vokal yang dihasilkan tidak sesuai dengan yang diinginkannya. Ia memulai lagi dari awal.

Model yang ke-dua mengantisipasi kebutuhan untuk memiliki alat-alat resonan yang bervariasi untuk berbagai macam suara yang diperlukan. Versi ini bersifat modular karena memiliki suatu ansambel yang terdiri dari tiga belas resonator, masing-masing memiliki buluh musikalnya sendiri, dan dapat dipindah-pindahkan sehingga kombinasi resonator yang berbeda dapat dicoba. Masing-masing resonator tersebut dilekatkan kepada salah satu dari tiga belas saluran yang tersedia pada bagian induk mesin. Selain itu, pada mesin ini terdapat tiga belas tuts, sama seperti yang terdapat pada sebuah piano, yang dapat ditekan untuk mendorong tekanan udara ke dalam salah satu saluran, menyebabkan buluh dari saluran tersebut bergetar. Dengan mesin ini, von Kempelen mengklaim telah berhasil menghasilkan versi yang “cukup bagus” dari vokal a, o, dan u, dan suara-suara yang “lumayan” untuk konsonan p, m, dan l.

Akan tetapi, mesin ini masih memiliki dua kesalahan besar. Pertama, pada bagian akhir dari suara-suara vokal cenderung mengandung tambahan suara seperti konsonan k. Kesalahan yang ke-dua ialah bila tuts-tuts yang ada ditekan, suara yang dihasilkan terdengar terpisah dan tidak mengalir bersama-sama dengan halus seperti pada bahasa alami. Untuk mengatasi kesalahan yang pertama, ia melapisi buluh-buluh dengan kulit yang halus, setelah menyadari bahwa penggunaan satu buluh tunggal, dibanding dengan serangkaian buluh, diperlukan untuk memperoleh paduan suara yang halus. Solusi ini memerlukan suatu proses disain ulang yang menyeluruh terhadap mesin ke-dua ini. Kembali, ia memulai lagi dari awal.



Gambar 2.2. Mesin Wolfgang von Kempelen Ke-Dua
Sumber: Pelton, G.E.(1996). Voice Processing. New York: McGraw-Hill.



Gambar 2.3. Mesin Wolfgang von Kempelen Ke-Tiga
Sumber: Pelton, G.E.(1996). Voice Processing. New York: McGraw-Hill.

Setiap mesin yang diciptakan von Kempelen harus "dimainkan" seperti sebuah instrumen musik. Mesin kreasinya yang ke-tiga dan terakhir berbeda dari yang ke-dua dalam hal penampilan, seperti yang nampak pada Gambar 2.3. di atas. Bagian tengah berfungsi seperti paru-paru manusia (dinamakan "*lungs*"),

diciptakan dengan secara terus-menerus memompakan pengembus naik turun menggunakan siku tangan kanan. Vokal dapat diproduksi dengan menutup "*nostrils*" dari mesin menggunakan telapak tangan kanan, memompa "*lungs*" menggunakan siku tangan kanan, dan menginterupsi karakteristik resonan dari suatu hubungan yang berbentuk seperti lonceng menggunakan telapak tangan kiri. Produksi setiap suara vokal dapat dilakukan dengan baik dan konstan bila si operator telah menguasai cara pengoperasian alat tersebut, tentu saja melalui banyak latihan. Sedangkan produksi suara konsonan membutuhkan lebih banyak latihan lagi.

Wolfgang von Kempelen mengklaim mesin ini dapat memproduksi suara semua vokal dan sembilan belas konsonan. Meskipun mesin von Kempelen yang ke-tiga ini dapat memompa enam kali lipat jumlah udara yang terdapat dalam paru-paru manusia, mesin ini hanya mampu mengucapkan beberapa frasa pendek sebelum kehabisan udara. Setiap suara memiliki konfigurasi ibu jari, telapak tangan, lengan, dan siku tangan yang berbeda.

2.3. Teknologi *Text-To-Speech*

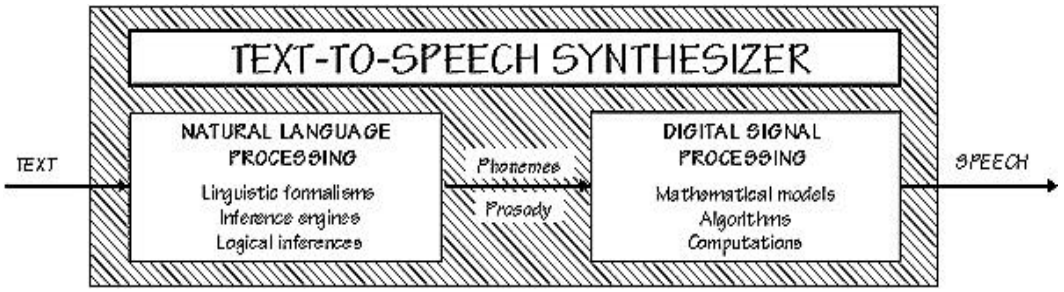
Sebuah sistem sintesis *text-to-speech* adalah suatu sistem berbasis komputer yang dapat membaca semua input teks, baik yang diinputkan kepada komputer oleh seorang operator maupun yang merupakan hasil *scan* dan dimasukkan ke dalam sebuah sistem OCR (*Optical Character Recognition*).

Suatu sistem sintesis *text-to-speech* secara bebas dapat disusun oleh 2 (dua) subsistem – analisa teks (sering disebut sebagai subsistem *Natural Language Processing*) dan penghasil suara percakapan (sering disebut sebagai subsistem *Digital Signal Processing*). Subsistem analisa teks mengkonversi input teks menjadi bentuk abstrak ketatabahasaan (fonem dan penekanan pengucapan) melalui struktur sintaktis dan fokus semantik dari suatu kalimat. Input teks pertama kali diproses oleh prosedur "Normalisasi Teks" yang memperluas setiap bentuk singkatan dan format non-teks menjadi bentuk rangkaian huruf yang dapat dibaca. Berdasarkan analisa secara semantik (arti kata), pragmatik (pengetahuan), dan sintaktik (struktural), penekanan-penekanan pada suatu kata ditambahkan dan fonem-fonem dikonversikan dari huruf-huruf. Prosedur penghasil suara

percakapan pertama-tama menggunakan struktur ketatabahasaan untuk menciptakan realisasi fonetik dari setiap fonem yang ada. (contoh: pemilihan alofon), garis bentuk frekuensi F0, dan motif durasi fonem. Prosedure penghasil suara percakapan kemudian melaksanakan transformasi *phonetic-to-accoustic* (atau biasa disebut sebagai sintesa suara percakapan). Transformasi *phonetic-to-accoustic* tersebut menghitung dan menyediakan berbagai parameter untuk melaksanakan sintesa suara percakapan.

Transformasi *phonetic-to-accoustic* untuk setiap fonem dalam suatu rangkaian fonetik tidak bersifat independen. Produksi suatu *phone* sangatlah dipengaruhi oleh tetangga-tetangganya (disebut sebagai koartikulasi). Jadi, menyimpan satu contoh *phone* untuk setiap fonem tidak akan dapat menghasilkan sintesa suara percakapan dengan kualitas yang baik.

Untuk dapat menghasilkan koartikulasi, terdapat 2 (dua) metode yang dapat dilaksanakan yaitu metode *rule-based* dan metode *concatenative.* Metode *rule-based* menyediakan informasi mengenai koartikulasi menggunakan aturan-aturan sintesa, sedangkan metode *concatenative* menyediakan informasi koartikulasi dalam suara percakapan yang telah disimpan sebelumnya yang dapat dikodekan sebagai parameter.



Gambar 2.4. Bagan Suatu Sistem Sintesa *Text-To-Speech*
Sumber: http://tcts.fpms.ac.be

Latar belakang dari lahirnya metode *concatenative* adalah bahwa aturan-aturan produksi dari fonem menjadi suara percakapan sangatlah rumit. Metode ini menghasilkan suara percakapan dengan menggabungkan bagian-bagian percakapan. Bagian-bagian percakapan dapat berupa suku kata, trifon atau difon. Informasi koartikulasi disimpan dalam bagian-bagian ini. Terdapat kurang lebih 10.000 suku kata dan 2.000 difon dalam bahasa Inggris.

2.4. Tata Bahasa Baku Bahasa Indonesia untuk Bahasa Lisan

Tata bahasa baku bahasa Indonesia untuk bahasa lisan mengatur berbagai hal yang berhubungan dengan komunikasi secara lisan menggunakan bahasa Indonesia. Berikut ini adalah beberapa hal yang perlu diperhatikan:

a. Fonem, merupakan satuan terkecil dari ciri-ciri bunyi bahasa yang membedakan arti. Dalam bahasa Indonesia, fonem dibagi menjadi 2 (dua) macam yaitu fonem segmental (fonem yang berwujud bunyi) dan fonem suprasegmental (fonem yang tidak berwujud bunyi, namun merupakan tambahan terhadap bunyi).
b. Suku kata, merupakan bagian kata yang diucapkan dalam 1 (satu) hembusan nafas yang umumnya terdiri dari beberapa fonem. Dalam bahasa Indonesia, suku kata dibedakan menjadi 2 (dua) macam, yaitu:
   - Suku buka, berakhir dengan vokal. Contoh: KV – “ku”.
   - Suku tutup, berakhir dengan konsonan. Contoh: KVK – “kan”.
c. Vokal, merupakan puncak dari suku kata. Dalam bahasa Indonesia dikenal 6 (enam) vokal, yaitu antara lain:
   - /i/ yang merupakan vokal tinggi depan.
   - /u/ yang merupakan vokal tinggi belakang.
   - /a/ yang merupakan vokal rendah tengah.
   - /o/ yang merupakan vokal sedang belakang, memiliki 2 (dua) kelompok pengucapan yaitu kelompok “orang”, “obat”, “protes” dan kelompok “bakso”, “soto”, “toko”.
   - /e/ yang merupakan vokal sedang depan, memiliki 2 (dua) kelompok pengucapan, yaitu kelompok “sore”, “kare” dan kelompok “perak”, “remeh”, “ejaan”.
   - /ê/ yang merupakan vokal sedang tengah. Contoh: “emas”, “bandeng”, “tipe”.
d. Diftong, merupakan 2 (dua) huruf vokal yang tidak dapat dipisahkan. Dalam bahasa Indonesia dikenal 3 (tiga) macam diftong, yaitu:
   - <au>, pada “harimau”.
   - <ai>, pada “pantai”, “sungai”.
   - <oi>, pada “amboi”.

e. Gugus konsonan, merupakan deretan 2 (dua) konsonan atau lebih yang tergolong dalam 1 (satu) suku kata yang sama. Dalam suatu suku kata, gugus konsonan dapat memiliki tempat di depan atau di belakang vokal atau diftong. Contoh: “praktik”, gugus konsonan /pr/ terletak di depan vokal /a/; “pers”, gugus konsonan /rs/ terletak di belakan vokal /e/.
f. Kalimat, merupakan rangkaian kata. Dalam bahasa Indonesia dikenal 3 (tiga) jenis kalimat utama, yaitu kalimat berita, kalimat tanya, dan kalimat perintah atau seru. Pada bahasa lisan, setiap jenis kalimat tersebut memiliki aturan pengucapan yang berbeda. Aturan pengucapan tersebut akan mempengaruhi suku kata-suku kata yang dikandung oleh kalimat tersebut, terutama dalam hal intonasi atau tinggi rendahnya nada pengucapan.

2.5. DFA (*Deterministic Finite Automaton*)

Finite Automaton memiliki konsep sebagai bentuk yang paling sederhana dari peralatan komputerisasi abstrak. Meskipun teori *finite automata* hanya berhubungan secara langsung dengan mesin-mesin sederhana, ini merupakan dasar penting dari banyak aplikasi baik konkrit maupun abstrak. *Finite-state control* dari suatu *finite automaton* juga merupakan inti dari begitu banyak peralatan komputer yang kompleks, salah satu di antaranya yaitu mesin Turing.

Pengaplikasian *finite automata* dapat ditemukan pada algoritma-algoritma yang digunakan untuk pencocokan *string* pada perangkat lunak editor teks dan perangkat lunak pengecekan ejaan, serta dapat ditemui juga pada penganalisa sintaks yang digunakan oleh *assemblers* atau *compilers*. Meskipun *finite automata* pada umumnya dianggap sebagai peralatan komputer abstrak, ia juga banyak ditemui pada aplikasi-aplikasi non-komputer, seperti pengatur sinyal lampu lalu-lintas dan mesin *vending*.

Secara singkat, sebuah ***deterministic finite automaton*** (**DFA**), juga disebut sebagai sebuah *recognizer* atau *acceptor*, adalah model matematis dari suatu peralatan komputer yang menggunakan *finite-state* yang dapat mengenali atau menerima suatu rangkaian kata (*words*) yang terdiri dari beberapa alfabet (*alphabet*), sedangkan rangkaian kata yang dikenali disebut sebagai bahasa (*language*). *Automaton* memiliki satu alur khusus dan unik untuk setiap kata yang

akan dikenali atau diterima, Jika suatu alur berakhir pada suatu *state* yang disebut sebagai *final state* atau *accepting state*, maka kata yang ditelusuri tersebut dikatakan dikenali oleh *automaton*.

Komponen dasar yang dimiliki oleh DFA adalah *alphabet* yaitu himpunan simbol/lambang yang dikenali. Himpunan alfabet diwakili dengan **å**, jika dan hanya jika ∑ merupakan himpunan simbol yang bersifat tetap dan bukan merupakan himpunan kosong. Contoh umum dari *alphabet* di sini adalah 26 (dua puluh enam) huruf yang dikenali dalam bahasa Inggris ataupun rangkaian karakter ASCII, yang merupakan rangkaian standar dari kode-kode komputer. Sedangkan sebuah *words* adalah rangkaian satu atau lebih *alphabet* yang telah dinyatakan sebelumnya. Berikut ini adalah contoh *alphabet* beserta *words* yang dapat dibentuknya:

- **å = {a, b}**, maka contoh *words* yang dapat dibentuknya yaitu **"aab"**, **"abab"**, **"a"**, **"bbbbbb"**, dan lain-lain.
- **å = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}**, maka contoh *words* yang dapat dibentuknya yaitu **"26498098"**, **"100103"**, **"0000"**, dan lain-lain.

Lebih lanjut, *concatenation* adalah proses menggabungkan dua buah *words* menjadi satu *words* baru, yaitu yang terdiri dari rangkaian *alphabet* dari *words* pertama dan disambung dengan rangkaian *alphabet* dari *words* ke-dua.

- **å = {a, b}**, *words* **x = "aaa"** dan **y = "bbb"** dimana setiap a∈∑ dan setiap b∈∑. Maka gabungan atau *concatenation* x dan y, dinyatakan dengan **x.y = "aaabbb"**.

Setelah memiliki pemahaman di atas, maka definisi dari sebuah DFA dapat ditetapkan sebagai suatu model matematis dari sebuah mesin yang menerima suatu rangkaian *words* tertentu yang mengandung *alphabet* ∑. DFA memiliki lima komponen, yaitu antara lain:

a. **å**, merupakan himpunan *alphabet* input (himpunan simbol/lambang yang tetap dan bukan merupakan himpunan kosong)
b. **S**, merupakan himpunan *state* yang tetap dan bukan merupakan himpunan kosong.
c. $\mathbf{S_0}$, merupakan *state* awal (*start state* atau *initial state*), merupakan anggota dari **S**.

d. **d**, merupakan fungsi transisi antar *state*; $\delta: S \times \sum \rightarrow S$.

e. **F**, merupakan himpunan *state* akhir (*final state* atau *accepting state*), merupakan sub-himpunan dari S.

Secara visual, suatu bagan DFA diwakili dengan suatu graf berarah dengan rumus G=<V,E>; dimana V=S dan E={<s,t,**a**> | s,t∈S, **a**∈ $\sum$ ∧ δ(s,**a**) = t}. "V" merupakan himpunan verteks pada graf, "E" merupakan himpunan sisi pada graf yang pada dasarnya merupakan fungsi-fungsi transisi antara *state* yang satu ke *state* yang lain (*state* "s" dan "t", yang masing-masingnya merupakan anggota dari "S"). Selain itu, setiap sisi graf diberi nama dengan *alphabet* penghubung (*alphabet* "**a**") antara dua verteks yang dihubungkannya.

Pada umumnya, dalam suatu bagan DFA terdapat minimal satu *state* akhir. Verteks graf yang menunjukkan suatu *state*, tetapi bukan *state* akhir, dinyatakan dengan lingkaran ◯ , sedangkan yang menunjukkan suatu *state* akhir dinyatakan dengan lingkaran ganda ◎ , sisi graf yang menunjukkan fungsi transisi dinyatakan dengan tanda panah ⟶ .

Jadi suatu *state* dapat menjadi asal dan tujuan dalam suatu fungsi transisi yang melibatkan dua buah *state*. Ditinjau dari sudut pandang *state* asal, maka setiap *state* (kecuali *state* akhir) pasti menjadi *state* asal dan memiliki fungsi transisi ke *state* yang lain, sedangkan *state* akhir dapat tidak memiliki fungsi transisi ke *state* yang lain. Ditinjau dari sudut pandang *state* tujuan, maka setiap *state* (kecuali *state* awal) pasti menjadi *state* tujuan.

Pada suatu *finite automaton*, suatu *state* dapat memiliki tujuan ke beberapa *state* yang berbeda dengan *alphabet* penghubung yang sama. Akan tetapi, hal ini tidak diperbolehkan pada suatu DFA. Untuk menyederhanakan suatu *finite automaton* menjadi suatu DFA dipergunakan Tabel Transisi yang memiliki kolom berupa variasi *alphabet* yang diterima dan baris berupa nama-nama *state* asal. Sedangkan titik temu antara suatu kolom dan baris diisi dengan nama-nama *state* tujuan dari *state* asal yang tertera pada bagian baris dengan *alphabet* penghubung yang tertera pada bagian kolom. Berikut ini adalah contoh suatu DFA yang akan mengenali suatu bilangan cacah.

- DFA Bilangan Cacah = <∑, S, S_0, F>

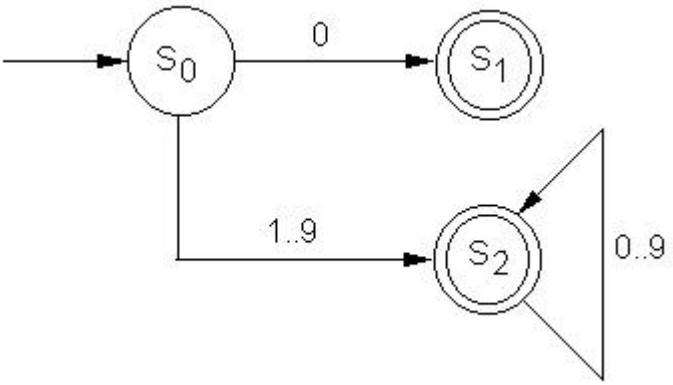
  ∑ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

  S = {S_0, S_1, S_2}

  S_0 = S_0

  F = {S_1, S_2}

  Berikut ini adalah bagan DFA untuk Bilangan Cacah:



Gambar 2.5. Bagan DFA Bilangan Cacah

Berikut ini adalah tabel transisi DFA untuk Bilangan Cacah:

Tabel 2.1. Tabel Transisi DFA Bilangan Cacah

d	**0**	**1..9**
S_0	S_1	S_2
S_1	-	-
S_2	S_2	S_2

Keterangan:

- Bagan pada Gambar 2.5. di atas sudah merupakan bagan DFA yang benar karena tidak ada *state* asal yang memiliki tujuan ke lebih dari satu *state* tujuan dengan *alphabet* penghubungan yang sama.

2.6. DirectSound dan DirectMusic pada Microsoft DirectX 8.1

Versi-versi terdahulu dari DirectX menampilkan fitur-fitur audio sebagai dua komponen yang berlainan dan terpisah: DirectSound dan DirectMusic. DirectSound digunakan untuk memainkan dan menangkap *sample-sample* digital

yang telah direkam sebelumnya (*waves*), dan DirectMusic digunakan untuk memainkan data berbasis *message* (*message-based* data) yang bervariasi dari file-file MIDI yang sederhana sampai dengan segmen-segmen musikal yang dibuat dengan DirectMusic Producer. Meskipun DirectMusic selalu dapat memainkan efek-efek suara non-musikal menggunakan DLS –singkatan dari *Downloadable Sound*, yaitu suatu aturan standar yang digunakan untuk melakukan sintesis suara-suara *wave* yang berasal dari *sample-sample* digital yang disimpan dalam suatu perangkat lunak. Aturan-aturan standar level 1 dan 2 DLS dipublikasikan oleh MIDI Manufacturers Association–, fungsi utamanya pada waktu itu adalah kemampuannya untuk memainkan musik.

Pada DirectX 8, *interface-interface* DirectMusic merupakan mekanisme utama untuk memuat (*loading*) dan memainkan semua suara, baik yang berasal dari file-file atau *resource-resource* dalam format *wave*, format MIDI, format DirectMusic Producer, maupun format-format lain yang dapat dikenali menggunakan *add-on loader* dan perangkat-perangkat yang telah disediakan.

Aplikasi-aplikasi masih dapat menggunakan *interface-interface* DirectSound untuk memainkan *wave*, dan DirectSoundCapture masih merupakan API untuk perekaman *wave*. Akan tetapi, pada banyak aplikasi DirectSound melaksanakan tugasnya pada bagian *downstream* dari *synthesizer* DirectMusic. DirectSound mengambil output dari *synthesizer*, menyalurkannya ke filter-filter efek, menambahkan efek-efek 3-D, dan melakukan proses *mixing* akhir sebelum mengalirkan data tersebut ke alat output.

Dibandingkan dengan DirectSound, *interface-interface* DirectMusic memberikan beberapa keuntungan sebagai berikut, untuk memuat (*loading*) dan memainkan *waves*:

- Tidak perlu melakukan pemilahan (*parsing*) terhadap suatu file atau *resource*, karena hal ini akan dilakukan oleh DirectMusic *loader*.
- Penggunaan *audio compression manager* (ACM) pada Windows secara otomatis, untuk format-format *wave* selain yang telah dikenali oleh DirectSound.

- Pengaliran data (data *streaming*) secara otomatis. Tidak perlu lagi melakukan pembuatan *buffer* dan penelusuran pointer-pointer baca dan tulis (*read and write pointers*) secara manual.
- Pengaturan waktu (*timing*) yang lebih baik. Tidak hanya lebih tepat dalam melakukan penjadwalan permainan suara, namun juga lebih mudah dalam melakukan sinkronisasi antara efek-efek suara dan musik.
- Pengaturan terhadap banyak proses yang melibatkan satu suara dengan lebih mudah.

Dengan adanya keuntungan-keuntungan tersebut, bukan berarti harus kehilangan akses terhadap kontrol-kontrol *low level* yang ditawarkan oleh DirectSound. DirectMusic API memungkinkan untuk mengakses setiap *object* yang terdapat dalam jalur audio (*audiopath*). Selain itu, fitur-fitur baru DirectSound API memungkinkan untuk menambahkan berbagai *filter* dan efek terhadap data audio.

2.6.1. Sekilas tentang Aliran Data Audio

Pada umumnya, sebuah program aplikasi yang menggunakan DirectSound atau DirectMusic memperoleh data musikal dari satu atau lebih sumber sebagai berikut:

- File-file MIDI.
- File-file *Wave.*
- File-file tersegmentasi yang dibuat dengan DirectMusic Producer atau aplikasi lain yang sejenis.
- File-file komponen yang dibuat dengan DirectMusic Producer dan digabung menjadi sebuah komposisi lengkap oleh suatu objek komposer.

Data dari sumber-sumber tersebut tersimpan dalam objek-objek segmen. Setiap objek segmen merepresentasikan data dari sebuah sumber tertentu. Pada suatu saat tertentu dalam sebuah *performance*, sebuah segmen primer dan sejumlah segmen sekunder dapat dimainkan bersama-sama. File-file sumber dapat dicampur dan dipadukan – misalnya, sebuah segmen sekunder yang berbasis pada suatu file *wave* dapat dimainkan bersama-sama dengan sebuah segmen primer yang berbasis pada suatu file segmen buatan.

Sebuah segmen terdiri dari satu atau lebih *track*, yang masing-masingnya mengandung data-data berskala waktu dari suatu jenis tertentu – misalnya perubahan not atau tempo. Pada saat sebuah segmen dimainkan, hampir semua *track* yang ada membentuk pesan-pesan berbasiskan waktu. Sedangkan jenis-jenis *track* yang lain menyediakan data hanya ketika diminta oleh *performance*.

Pertama-tama, *performance* mengirim pesan-pesan kepada *tool-tool* yang telah ditentukan aplikasi. *Tool-tool* tersebut dikelompokkan ke dalam *segment toolgraphs* yang hanya memproses pesan-pesan dari segmen-segmen tertentu, *audiopath tollgraphs* untuk pesan-pesan dari semua segmen yang sedang dimainkan pada *path*, dan sebuah *performance toolgraph* yang menerima pesan dari semua segmen. Sebuah *tool* dapat memodifikasi suatu pesan dan meneruskannya, menghapusnya, atau mengirimkan sebuah pesan baru.

Dan akhirnya pesan-pesan dikirimkan kepada *tool* output, yang mengkonversi data menjadi format MIDI sebelum meneruskannya kepada *synthesizer*. Pesan-pesan MIDI yang berisi nomor-nomor *channel* spesifik diteruskan ke kelompok *channel* terkait dalam *synthesizer*. Kemudian *synthesizer* membuat gelombang-gelombang suara (*sound waves*) dan mengalirkannya ke dalam suatu alat yang disebut *sink*, yang akan mengatur distribusi data melalui *bus-bus* ke *buffer-buffer* DirectSound.

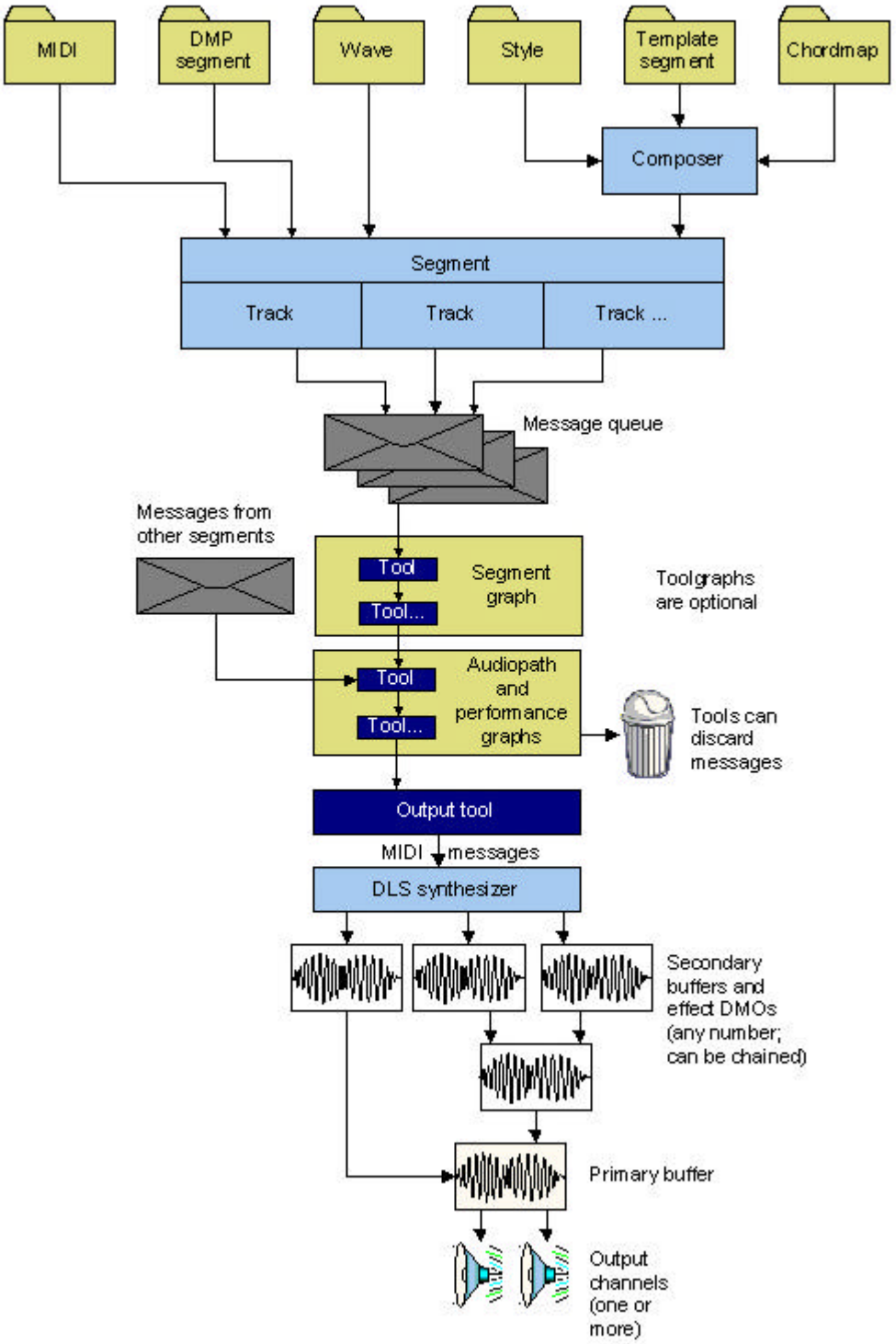
Berikut ini adalah tiga jenis *buffer* DirectSound:

1. *Sink-in buffers*, merupakan *buffer* sekunder DirectSound tempat *sink* mengalirkan data. *Buffer-buffer* jenis ini mengkonversi format data sesuai dengan format *buffer* primer, jika diperlukan, dan memungkinkan aplikasi untuk mengatur *pan*, volume, 3-D *spatialization*, dan *property-property* lainnya. *Buffer-buffer* jenis ini juga dapat meneruskan data yang mereka miliki melalui modul-modul efek untuk menambahkan efek-efek tertentu, seperti *reverberation* dan *echo*, dan kemudian *waveform* yang dihasilkan akan diteruskan baik ke *primary buffer* maupun satu atau lebih *mix-in buffers*.
2. *Mix-in buffers*, akan menerima data dari *buffer-buffer* yang lain, menambahkan efek-efek yang diinginkan, dan menghasilkan *waveform* yang baru. *Buffer-buffer* jenis ini dapat digunakan untuk mengaplikasikan efek global. Sebuah efek yang didapat dengan mengalirkan data ke dalam suatu

mix-in buffer disebut sebagai sebuah *send*. *Mix-in buffers* hanya dapat dibuat menggunakan konfigurasi-konfigurasi *audiopath* dalam DirectMusic Producer.

3. *Primary buffer*, akan melakukan proses pencampuran (*mixing*) akhir terhadap semua data dan kemudian meneruskannya ke *rendering device*.

Pada halaman 24 dari buku laporan ini adalah bagan yang memberikan gambaran sederhana aliran data dari file-file ke *speaker*. Pada bagan tersebut digunakan sebuah segmen, meskipun beberapa segmen sekaligus dapat dimainkan pada waktu yang sama. Segmen tersebut memperoleh data hanya dari satu dari keempat sumber yang ada, yaitu: sebuah file *wave*, sebuah file MIDI, sebuah file segmen yang dibuat dengan DirectMusic Producer, dan file-file komponen yang dikombinasikan oleh objek komposer.



Gambar 2.6. Bagan Aliran Data Audio
Sumber: Microsoft DirectX 8.1 Documentation for C++.

2.6.2. Memainkan File Audio

Berikut ini adalah empat langkah yang dilakukan untuk memainkan suatu file audio menggunakan DirectMusic, yang dijelaskan dengan contoh implementasi pemrograman dalam bahasa C++:

1. "Menginisialisasi".

   Include-include berikut ini diperlukan oleh setiap aplikasi yang menggunakan DirectMusic API. Melakukan *include* file *header* **Dmusici.h** juga akan menyebabkan file-file *header* lain yang diperlukan di-*include* secara otomatis.

```
#define INITGUID
#include <dmusici.h>
```

   Contoh ini menggunakan tiga *pointer interface*, yang dideklarasikan sebagai berikut:

```
IDirectMusicLoader8* g_pLoader = NULL;
IDirectMusicPerformance8* g_pPerformance = NULL;
IDirectMusicSegment8* g_pSegment = NULL;
```

   Semua kode pemrograman dalam contoh ini dituliskan di dalam *function* **WinMain**. Aplikasi yang dihasilkan oleh contoh ini tidak memiliki *window* utama, jadi dapat secara langsung dilakukan proses pembuatan COM dan dua objek utama, yaitu *loader* dan *performance*:

```
INT APIENTRY WinMain( HINSTANCE hInst, HINSTANCE
hPrevInst, LPSTR pCmdLine, INT nCmdShow )
{
    CoInitialize(NULL);

    CoCreateInstance(CLSID_DirectMusicLoader, NULL,
        CLSCTX_INPROC, IID_IDirectMusicLoader8,
        (void**)&g_pLoader);

    CoCreateInstance(CLSID_DirectMusicPerformance, NULL,
        CLSCTX_INPROC, IID_IDirectMusicPerformance8,
        (void**)&g_pPerformance );
```

Langkah selanjutnya adalah menginisialisasi *performance* dan *synthesizer*. *Method* **IDirectMusicPerformance8::InitAudio** melakukan fungsi-fungsi sebagai berikut:

- Membuat sebuah objek DirectMusic dan sebuah objek DirectSound. Pada kasus kebanyakan tidak diperlukan *interface* untuk kedua objek di atas, oleh karena itulah dapat diisikan nilai NULL untuk dua parameter pertama.
- Menghubungkan sebuah *window* aplikasi dengan objek DirectSound. Secara normal *handle* dari *window* aplikasi utama diisikan sebagai parameter ke-tiga, namun dalam aplikasi pada contoh ini tidak memiliki *window*, jadi diisikan nilai NULL.
- Menyediakan suatu *audiopath* umum dari sebuah tipe standar. Pada contoh ini digunakan *path* dari tipe **DMUS_APATH_SHARED_STEREOPLUSREVERB**, yang sesuai untuk musik.
- Mengalokasikan sejumlah *performance channel* kepada *audiopath.* File-file *wave* membutuhkan sebuah *performance channel* tunggal, dan file-file MIDI membutuhkan sampai 16 buah. Segmen-segmen yang dibuat dalam DirectMusic Producer mungkin membutuhkan lebih. Tidak akan menimbulkan kesalahan jika mengalokasikan *channel* secara berlebih.
- Menentukan kemampuan dan sumber daya yang dimiliki *synthesizer*. Hal ini dapat dilakukan dengan salah satu dari dua cara yang ada: dengan mengeset *flags* atau dengan menyediakan suatu struktur **DMUS_AUDIOPARAMS** disertai informasi yang lebih mendetail. Kebanyakan aplikasi mengeset *flag* **DMUS_AUDIOF_ALL** dan membiarkan DirectMusic membuat *synthesizer* dengan parameter-parameter umum.

Pada contoh ini pemanggilan InitAudio dilakukan dengan sangat sederhana:

```
g_pPerformance->InitAudio(
    NULL,      // IDirectMusic interface not needed.
    NULL,      // IDirectSound interface not needed.
    NULL,      // Window handle.
```

```
    DMUS_APATH_SHARED_STEREOPLUSREVERB,  //
Default    audiopath type.
    64,        // Number of performance channels.
    DMUS_AUDIOF_ALL,   // Features on synthesizer.
    NULL       // Audio parameters; use defaults.
);
```

2. “Memuat sebuah file”.

Performance dan *synthesizer* DirectMusic sekarang siap untuk memproses data suara. Untuk memperoleh data tersebut, *loader* perlu mengetahui dimana menemukannya. Meskipun sebuah *path* lengkap dapat disediakan setiap kali sebuah file dimuat, adalah hal yang lebih efisien untuk menetapkan suatu direktori standar. Hal ini dilakukan dengan memanggil *method* **IDirectMusicLoader8::SetSearchDirectory**. Pada contoh kode pemrograman ini, *path* standar yang diberikan adalah direktori media milik Windows.

```
// Find the Windows media directory.

CHAR strPath[MAX_PATH];
GetWindowsDirectory( strPath, MAX_PATH );
strcat( strPath, "\\media" );

// Convert to Unicode.

WCHAR wstrSearchPath[MAX_PATH];
MultiByteToWideChar( CP_ACP, 0, strPath, -1,
      wstrSearchPath, MAX_PATH );

// Set the search directory.

g_pLoader->SetSearchDirectory(
  GUID_DirectMusicAllTypes, // Types of files sought.
  wstrSearchPath,     // Where to look.
  FALSE         // Don't clear object data.
);
```

Pada pemanggilan **SetSearchDirectory**, parameter ***fClear*** diisi dengan nilai FALSE karena tidak ada bahaya yang timbul bila secara tidak sengaja terjadi pemuatan ulang objek dari direktori yang salah. Hal ini mungkin terjadi hanya bila aplikasi memuat objek-objek yang memiliki nama yang sama persis dari *folder-folder* yang berbeda.

Sekarang setelah *loader* mengetahui dimana menemukan file yang dimaksud, ia dapat memuat file terkait sebagai suatu segmen:

```
WCHAR wstrFileName[MAX_PATH] = L"The Microsoft
Sound.wav";

if (FAILED(g_pLoader->LoadObjectFromFile(
  CLSID_DirectMusicSegment, // Class identifier.
  IID_IDirectMusicSegment8, // ID of desired interface.
  wstrFileName,     // Filename.
  (LPVOID*) &g_pSegment   // Pointer that receives interface.
)))
{
  MessageBox( NULL, "Media not found, sample will now quit.",
       "DMusic Tutorial", MB_OK );
  return 0;
}
```

3. “Memainkan audio”.

File *wave* yang telah dimuat dalam langkah sebelumnya telah tersedia bagi *performance* melalui *interface* **IDirectMusicSegment8**. Sebelum suatu segmen dapat dimainkan, *band*-nya harus di-*download* ke *synthesizer*. Selama *band* ini tidak di-*unload*, maka langkah ini hanya dilakukan satu kali untuk setiap segmen yang menggunakan suatu *band* tertentu.

Kode pemrograman berikut, yang berasal dari *function* **WinMain**, men-*download band* ke *performance*. Sebagai alternatif, *band* tersebut dapat di-*download* ke sebuah *audiopath*. Selama hanya terdapat satu *synthesizer* tunggal yang digunakan, tidak perlu dipermasalahkan objek tujuan mana yang dipilih:

```
g_pSegment->Download( g_pPerformance );
```

Hal yang harus dilakukan untuk memainkan file terkait adalah meneruskan *interface* segmen ke **IDirectMusicPerformance8::PlaySegmentEx**. *Method* ini menyediakan banyak pilihan untuk memainkan suara, namun untuk memainkan sebuah segmen secara seketika pada *audiopath* standar, semua parameter kecuali parameter pertama dapat diisi dengan NULL atau 0.

```
g_pPerformance->PlaySegmentEx(
    g_pSegment,  // Segment to play.
    NULL,    // Used for songs; not implemented.
    NULL,    // For transitions.
    0,     // Flags.
    0,     // Start time; 0 is immediate.
    NULL,    // Pointer that receives segment state.
    NULL,    // Object to stop.
    NULL   // Audiopath, if not default.
);
MessageBox( NULL, "Click OK to Exit.", "Play Audio", MB_OK
);
```

4. “Menutup”.

Untuk mengakhiri sebuah aplikasi audio dengan “bersih”, terdapat empat langkah utama yang harus dilakukan:

- Menghentikan setiap segment yang sedang dimainkan dengan memanggil **IDirectMusicPerformance8::Stop**.
- Menutup *performance*. *Method* **IDirectMusicPerformance8::CloseDown** melakukan berbagai tugas pembersihan dan melepaskan berbagai referensi internal yang digunakan oleh berbagai objek yang ada.
- Melepaskan semua *interface*.
- Menutup COM.

Contoh kode pemrograman berikut ini termasuk di dalam *function* **WinMain** dan akan dipanggil bila *dialog box* ditutup.

```
g_pPerformance->Stop(
    NULL, // Stop all segments.
    NULL, // Stop all segment states.
    0,  // Do it immediately.
    0   // Flags.
  );

  g_pPerformance->CloseDown();
  g_pLoader->Release();
  g_pPerformance->Release();
  g_pSegment->Release();

  CoUninitialize();

  return 0;  // Return value for WinMain.
}
```

2.7. Format File WAVE

Format file WAVE adalah sebuah turunan dari spesifikasi RIFF yang diciptakan Microsoft untuk menyimpan berbagai file *multimedia*. Sebuah file RIFF dimulai dengan sebuah *header* file yang diikuti dengan deretan data. Sebuah file WAVE seringkali hanya merupakan suatu file RIFF dengan bagian "WAVE" yang terdiri dari dua kelompok, yaitu kelompok "fmt" yang mendefinisikan format data dan kelompok "data" yang berisi *sample* data itu sendiri.

Berikut ini adalah struktur sebuah file dengan format WAVE yang disusun dalam sebuah tabel.

Tabel 2.2 Struktur File dengan Format WAVE

	Offset	Ukuran (byte)	Nama	Keterangan
Header File RIFF	0	4	ChunkID	"RIFF"
	4	4	ChunkSize	36 + SubChunk2Size
	8	4	Format	"WAVE"
Bagian WAVE, Kelompok "fmt"	12	4	SubChunk1ID	"fmt"
	16	4	SubChunk1Size	16 (untuk PCM), merupakan ukuran total kelompok "fmt"
	20	2	AudioFormat	1 (untuk PCM)
	22	2	NumChannels	Mono = 1, Stereo = 2
	24	4	SampleRate	8000, 22050, 44100, dst.
	28	4	ByteRate	= SampleRate * NumChannels * BitsPerSample/8
	32	2	BlockAlign	= NumChannels * BitsPerSample/8
	34	2	BitsPerSample	8 bits = 8, 16 bits = 16, dst.
Bagian WAVE, Kelompok "data"	36	4	SubChunk2ID	"data"
	40	4	SubChunk2Size	= NumSamples * NumChannels * BitsPerSample/8
	44	*	Data	*sample* data