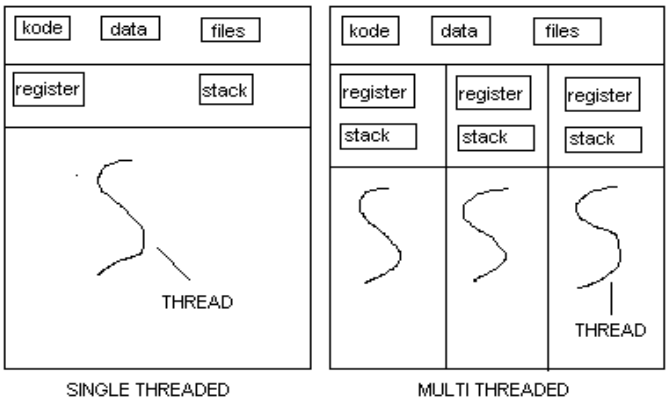
Bab 3. Proses

I. Thread

Thread adalah sebuah alur kontrol dari sebuah proses. Kontrol *thread* tunggal ini hanya memungkinkan proses untuk menjalankan satu tugas pada satu waktu. Banyak sistem operasi modern telah memiliki konsep yang dikembangkan agar memungkinkan sebuah proses untuk memiliki eksekusi *multi-threads*, agar dapat secara terus menerus mengetik dan menjalankan pemeriksaan ejaan didalam proses yang sama, maka sistem operasi tersebut memungkinkan proses untuk menjalankan lebih dari satu tugas pada satu waktu. Suatu proses yang multithreaded mengandung beberapa perbedaan alur kontrol dengan ruang alamat yang sama.

Keuntungan dari multithreaded meliputi peningkatan respon dari pengguna, pembagian sumber daya proses, ekonomis, dan kemampuan untuk mengambil keuntungan dari arsitektur multiprosesor. *Thread* merupakan unit dasar dari penggunaan CPU, yang terdiri dari Thread_ID, *program counter*, *register set*, dan *stack*. Sebuah *thread* berbagi *code section*, *data section*, dan sumber daya sistem operasi dengan Thread lain yang dimiliki oleh proses yang sama. Thread juga sering disebut *lightweight process*. Sebuah proses tradisional atau *heavyweight process* mempunyai *thread* tunggal yang berfungsi sebagai pengendali.

Perbedaan antara proses dengan *thread* tunggal dengan proses dengan *thread* yang banyak adalah proses dengan *thread* yang banyak dapat mengerjakan lebih dari satu tugas pada satu satuan waktu.



Gambar 3.1. Thread

Banyak perangkat lunak yang berjalan pada PC modern dirancang secara *multi-threading*. Sebuah aplikasi biasanya diimplementasi sebagai proses yang terpisah dengan beberapa *thread* yang berfungsi sebagai pengendali. Contohnya sebuah *web browser* mempunyai *thread* untuk menampilkan gambar atau tulisan sedangkan *thread* yang lain berfungsi sebagai penerima data dari network.

Kadang kala ada situasi dimana sebuah aplikasi diperlukan untuk menjalankan beberapa tugas yang serupa. Sebagai contohnya sebuah *web server* dapat mempunyai ratusan klien yang mengaksesnya secara *concurrent*. Kalau *web server* berjalan sebagai proses yang hanya mempunyai *thread* tunggal maka ia hanya dapat melayani satu klien pada pada satu satuan waktu. Bila ada klien lain yang ingin mengajukan permintaan maka ia harus menunggu sampai klien sebelumnya selesai dilayani. Solusinya adalah dengan membuat *web server* menjadi *multi-threading*. Dengan ini maka sebuah *web server* akan membuat *thread* yang akan mendengar permintaan klien, ketika permintaan lain diajukan maka *web server* akan menciptakan *thread* lain yang akan melayani permintaan tersebut.

Keuntungan *Thread*

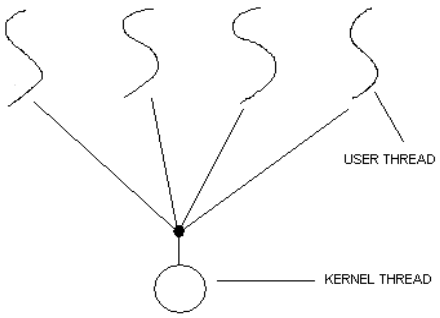
Keuntungan dari program yang *multithreading* dapat dipisah menjadi empat kategori:

1. **Responsi**: Membuat aplikasi yang interaktif menjadi *multithreading* dapat membuat sebuah program terus berjalan meski pun sebagian dari program tersebut diblok atau melakukan operasi yang panjang, karena itu dapat meningkatkan respons kepada pengguna. Sebagai contohnya dalam *web browser* yang *multithreading*, sebuah *thread* dapat melayani permintaan pengguna sementara *thread* lain berusaha menampilkan image.
2. **Berbagi sumber daya**: *thread* berbagi memori dan sumber daya dengan *thread* lain yang dimiliki oleh proses yang sama. Keuntungan dari berbagi kode adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa *thread* yang berbeda dalam lokasi memori yang sama.
3. **Ekonomi**: dalam pembuatan sebuah proses banyak dibutuhkan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan penggunaan *thread*, karena *thread* berbagi memori dan sumber daya proses yang memilikinya maka akan lebih ekonomis untuk membuat dan *context switch thread*. Akan susah untuk mengukur perbedaan waktu antara proses dan *thread* dalam hal pembuatan dan pengaturan, tetapi secara umum pembuatan dan pengaturan proses lebih lama dibandingkan *thread*. Pada Solaris, pembuatan proses lebih lama 30 kali dibandingkan pembuatan *thread*, dan *context switch* proses 5 kali lebih lama dibandingkan *context switch thread*.
4. **Utilisasi arsitektur *multiprocessor***: Keuntungan dari multithreading dapat sangat meningkat pada arsitektur *multiprocessor*, dimana setiap *thread* dapat berjalan secara pararel di atas processor yang berbeda. Pada arsitektur processor tunggal, CPU menjalankan setiap *thread* secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi pararel, tetapi pada kenyataannya hanya satu *thread* yang dijalankan CPU pada satu-satuan waktu (satu-satuan waktu pada CPU biasa disebut *time slice* atau *quantum*).

Multithreading Models

- **Many-to-One Model**

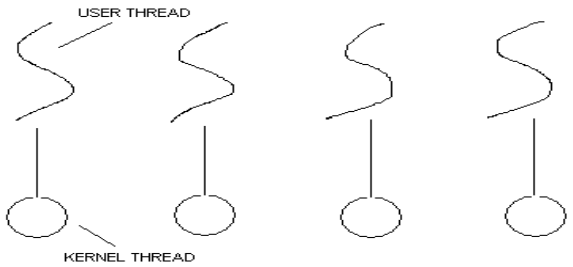
Many-to-One model memetakan banyak user-level thread ke satu kernel thread. Pengaturan thread dilakukan di user space, oleh karena itu ia efisien tetapi ia mempunyai kelemahan yang sama dengan user thread. Selain itu karena hanya satu thread yang bisa mengakses thread pada suatu waktu maka *multiple* thread tidak bisa berjalan secara paralel pada *multiprocessor* . User-level thread yang diimplementasi pada sistem operasi yang tidak mendukung kernel thread menggunakan Many-to-One model.



Gambar 3.2. Many-To-One

❖ **One-to-One Model**

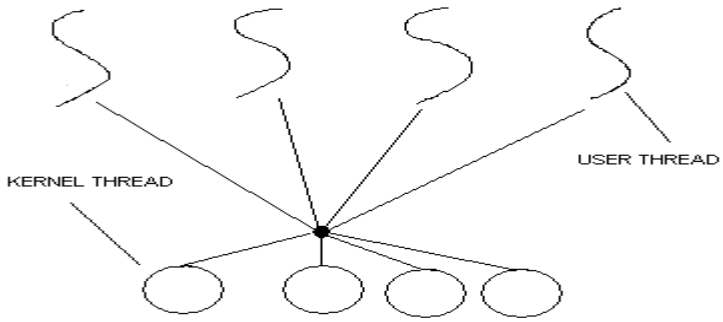
One-to-One model memetakan setiap user thread ke kernel thread. Ia menyediakan lebih banyak *concurrency* dibandingkan Many-to-One model. Keuntungannya sama dengan keuntungan kernel thread. Kelemahannya model ini adalah setiap pembuatan user thread membutuhkan pembuatan kernel thread. Karena pembuatan thread bisa menurunkan performa dari sebuah aplikasi maka implmentasi dari model ini membatasi jumlah thread yang dibatasi oleh sistem. Contoh sistem operasi yang mendukung One-to-One model adalah Windows NT dan OS/2.



Gambar 3.3. One-To-One

❖ **Many-to-Many Model**

Many-to-many model *multiplexes* banyak user-level thread ke kernel thread yang jumlahnya lebih kecil atau sama banyaknya dengan user-level thread. Jumlah kernel thread bisa spesifik untuk sebagian aplikasi atau sebagian mesin. Many-to-One model mengijinkan developer untuk membuat user thread sebanyak yang ia mau tetapi *concurrency* tidak dapat diperoleh karena hanya satu thread yang bisa dijadwal oleh kernel pada suatu waktu. One-to-One menghasilkan concurrency yang lebih tetapi developer harus hati-hati untuk tidak menciptakan terlalu banyak thread dalam suatu aplikasi(dalam beberapa hal, developer hanya bisa membuat thread dalam jumlah yang terbatas). Many-to-Many model tidak menderita kelemahan dari 2 model di atas. Developer bisa membuat user thread sebanyak yang diperlukan, dan kernel thread yang bersangkutan bisa bejalan secara pararel pada *multiprocessor* . Dan juga ketika suatu thread menjalankan *blocking system call* maka kernel dapat menjadwalkan thread lain untuk melakukan eksekusi. Contoh sistem operasi yang mendukung model ini adalah Solaris, IRIX, dan Digital UNIX.



Gambar 3.4. Many-To-Many

Cancellation

Thread cancellation adalah tugas untuk memberhentikan thread sebelum ia menyelesaikan tugasnya. Sebagi contohnya jika dalam program java kita hendak mematikan JVM(Java Virtual Machine) maka sebelum JVM-nya dimatikan maka seluruh thread yang berjalan dihentikan terlebuh dahulu. Thread yang akan diberhentikan biasa disebut target thread.

Pemberhentian target thread bisa terjadi melalui 2 cara yang berbeda :

1. ***Asynchronous cancellation***: suatu thread seketika itu juga memberhentikan target thread.
2. ***Defered cancellation***: target thread secara perodik memeriksa apakah dia harus berhenti, cara ini memperbolehkan targetthread untuk memberhentikan dirinya sendiri secara terurut.

Hal yang sulit dari pemberhentian thread ini adalah ketika terjadi situasi dimana sumber daya sudah dialokasikan untuk thread yang akan diberhentikan. Selain itu kesulitan lain adalah ketika thread yang diberhentikan sedang meng-update data yang ia bagi dengan thread lain. Hal ini akan menjadi masalah yang sulit apabila digunakan *asynchronous cancellation* . Sistem operasi akan mengambil kembali sumber daya dari thread yang diberhentikan tetapi seringkali sistem operasi tidak mengambil kembali semua sumber daya dari thread yang diberhentikan.

Alternatifnya adalah dengan menggunakan *Deffered cancellation* . Cara kerja dari *deffered cancellation* adalah dengan menggunakan 1 thread yang berfungsi sebagai pengindikasi bahwa target thread hendak diberhentikan. Tetapi pemberhentian hanya akan terjadi jika target thread memeriksa apakah ia harus berhenti atau tidak. Hal ini memperbolehkan thread untuk memeriksa apakah ia harus berhenti pada waktu dimana ia bisa diberhentikan secara aman yang aman. Pthread merujuk tersebut sebagai *cancellation points* .

Pada umumnya sistem operasi memperbolehkan proses atau thread untuk diberhentikan secara *asynchronous* . Tetapi Pthread API menyediakan *deferred cancellation* . Hal ini berarti sistem operasi yang mengimplementasikan Pthread API akan mengijinkan *deferred cancellation* .

II. Client – Server

Menurut *Gallaugher & Ramanathan (1996)* :“client/server adalah client mengirim permintaan ke server, server menterjemahkan pesan, kemudian berusaha memenuhi permintaan”. Sedangkan menurut *Blaha & Premerlani (1998)* : “client/server adalah suatu arsitektur dimana sumber daya server menyediakan komputasi untuk banyak komponen client. Client dapat mengakses satu server atau multiple server. Client dan server bisa berjalan pada mesin yg sama atau berbeda, ditulis dalam berbagai bahasa dan menggunakan sistem operasi yang berbeda.

Secara umum Client/Server adalah arsitektur jaringan aplikasi yang memisahkan klien dari server (umumnya GUI). Setiap satuan perangkat lunak klien berhubungan dengan perangkat lunak server. Client/server adalah arsitektur berskala dimana setiap komputer atau proses pada jaringan berperan sebagai klien atau server. Piranti lunak server pada umumnya walaupun tidak selalu akan menjalankan komputer dengan kekuatan penuh yang secara khusus dan ekslusif menjalankan aplikasi bisnis. Sedangkan Piranti lunak Client pada umumnya berjalan pada PC/workstation biasa.

Client akan mendapatkan seluruh informasinya dan mengirimkannya kepada perangkat lunak server untuk sebuah keperluan, sebagai contoh konfigurasi file, kuota penyimpanan, program aplikasi bisnis atau untuk membebaskan intensifitas pekerjaan komputasi dan menkondisikan komputer Client bebas dan siap menjalan pekerjaan lainnya.

Client yang paling poluper dan digunakan secara luas saat ini adalah web browser dimana berkomunikasi dengan web server melalui internet untuk mendapatkan dan menampilkan isi halaman web.

Bentuk lain dari arsitektur client/server dikenal dengan istilah 'thin client', dimana terdapat sejumlah Client yang sedikit. Thin client menggunakan sedikit sumber dari PC Host yang ada. Tugas dari thin client pada umumnya hanya menampilkan secara grafik tentang informasi umum dari perangkat lunak server. Hal ini memungkinkan perusahaan untuk lebih mudah mengelola bisnis logicnya untuk semua aplikasi-aplikasi pada pusat lokasi.

Anatomi suatu program server

Peranan utama suatu program server adalah melayani client yang berjumlah banyak yang memiliki tujuan untuk menggunakan secara bersama sumber daya yang dimiliki oleh server tersebut. Berikut ini adalah karasteristik suatu yang biasanya dimiliki oleh suatu program server.

- **Menanti permintaan client**. Program server menghabiskan sebagian besar waktu kerjanya secara pasif menanti permintaan client. Biasanya permintaan ini datang dalam bentuk message melalui sesi komunikasi. Beberapa server menggunakan suatu sesi khusus untuk setiap client. Server yang lainnya menggunakan session yang digunakan secara dinamis. Ada juga yang menggunakan gabungan kedua teknik ini (dedicated dan dinamis). Untuk dapat bekerja dengan baik, server harus tetap dapat bekerja ketika terjadi permintaan yang banyak *(rush hour traffic)*.
- **Melaksanakan banyak permintaan pelayanan pada saat yang bersamaan**. Server harus sesegera mungkin melaksanakan pelayanan yang diminta oleh client. Jelas ini berarti, bahwa client tak boleh bergantung pada proses server yang hanya memiliki thread tunggal. Server harus dapat secara konkuren menyediakan pelayanan dengan tetap menjaga integritas sumber dayanya.
- **Mendahulukan client yang memiliki prioritas lebih tinggi (VIP)**. Server harus menyediakan beberapa tingkatan prioritas untuk clientnya. Misal untuk suatu pekerjaan batch dilakukan pada tingkatan prioritas yang rendah, sedangkan untuk pekerjaan yang berkaitan dengan *On Line Transaction Processing (OLTP)* dilakukan dengan prioritas tinggi.
- **Memulai dan melaksanakan aktifitas pekerjaan di background**. Server harus dapat menjalankan program di back ground, misal melakukan download record dari database utama selama waktu tidak sibuk. Inisiatif ini harus dapat dilakukan secara otomatis oleh server.
- **Tetap menjaga agar sistem tetap selalu bekerja**. Program server biasanya tergolong *mission-critical application.* Akan terjadi kerugian bila server tak bekerja melayani client. Dengan demikian program server dan environmentnya harus dapat bekerja secara robust (tahan terhadap gangguan).
- **Bertambah besar**. Biasanya program server membutuhkan memori dan prosesor yang besar. Environment dari server haruslah dapat di upgrade dan memiliki skalabilitas yang baik.

III. Agent

Software Agent adalah entitas perangkat lunak yang didedikasikan untuk tujuan tertentu yang memungkinkan user untuk mendelegasikan tugasnya secara mandiri, selanjutnya software agent nantinya disebut agent saja. Agen bisa memiliki ide sendiri mengenai bagaimana menyelesaikan suatu pekerjaan tertentu atau agenda tersendiri. Agen yang tidak berpindah ke host lain disebut *stationary agent.*

Definisi agen yang lebih rinci, ditinjau dari sudut pandang sistem, adalah obyek perangkat lunak yang:

1. Diletakan dalam lingkungan eksekusi
2. Memiliki sifat sebagai berikut :
    a. Reaktif, dapat merasakan perubahan dalam lingkungannya dan bertindak sesuai perubahan tersebut.
    b. *Autonomous*, mampu mengendalikan tindakannya sendiri
    c. Proaktif, mempunyai dorongan untuk mencapai tujuan
    d. Bekerja terus menerus sampai waktu tertentu

3. Dapat mempunyai sifat ortogonal sebagai berikut :
    a. Komunikatif, dapat berkomunikasi dengan agen yang lain.
    b. *Mobile* , dapat berpindah dari satu host ke host yang lain
    c. *Learning*, mampu menyesuaikan diri berdasarkan pengalaman sebelumnya
    d. Dapat dipercaya sehingga menimbulkan kepercayaan kepada *end user.*

Karakteristik dari Agen:

1. ***Autonomy***: Agent dapat melakukan tugas secara mandiri dan tidak dipengaruhi secara langsung oleh user, agent lain ataupun oleh lingkungan (environment). Untuk mencapai tujuan dalam melakukan tugasnya secara mandiri, agent harus memiliki kemampuan kontrol terhadap setiap aksi yang mereka perbuat, baik aksi keluar maupun kedalam [Woolridge et. al., 1995]. Dan satu hal yang penting yaitu mendukung autonomy dan masalah intelegensi (*intelligence*) dari agent.
2. ***Intelligence, Reasoning, dan Learning***: Setiap agent harus mempunyai standar minimum untuk bisa disebut agent, yaitu intelegensi (*intelligence*). Dalam konsep *intelligence*, ada tiga komponen yang harus dimiliki: internal *knowledge b*ase, kemampuan *reasoning* berdasar pada *knowledge base* yang dimiliki, dan kemampuan *learning* untuk beradaptasi dalam perubahan lingkungan.
3. ***Mobility dan Stationary***: Khusus untuk *mobile agent*, dia harus memiliki kemampuan yang merupakan karakteristik tertinggi yang dia miliki yaitu mobilitas. Berbeda dengan *stationary agent*. Tetapi keduanya tetap harus memiliki kemampuan untuk mengirim pesan dan berkomunikasi dengan agent lain.
4. ***Delegation***: Sesuai dengan namanya dan seperti yang sudah kita bahas pada bagian definisi, agent bergerak dalam kerangka menjalankan tugas yang diperintahkan oleh user. Fenomena pendelegasian (*delegation*) ini adalah karakteristik utama suatu program disebut agent.
5. ***Reactivity***: Karakteristik agent yang lain adalah kemampuan untuk bisa cepat beradaptasi dengan adanya perubahan informasi yang ada dalam suatu lingkungan (enviornment). Lingkungan itu bisa mencakup: agent lain, user, informasi dari luar, dsb [Brenner et. al., 1998].
6. ***Proactivity dan Goal-Oriented***: Sifat *proactivity* boleh dibilang adalah kelanjutan dari sifat *reactivity*. Agent tidak hanya dituntut bisa beradaptasi terhadap perubahan lingkungan, tetapi juga harus mengambil inisiatif langkah penyelesaian apa yang harus diambil [Brenner et. al., 1998]. Untuk itu agent harus didesain memiliki tujuan (*goal*) yang jelas, dan selalu berorientasi kepada tujuan yang diembannya (*goal-oriented*).
7. ***Communication and Coordination Capability***: Agent harus memiliki kemampuan berkomunikasi dengan user dan juga agent lain. Masalah komunikasi dengan user adalah masuk ke masalah user interface dan perangkatnya, sedangkan masalah komunikasi, koordinasi, dan kolaborasi dengan *agent* lain adalah masalah sentral penelitian *Multi Agent System* (MAS). Bagaimanapun juga, untuk bisa berkoordinasi dengan *agent* lain dalam menjalankan tugas, perlu bahasa standard untuk berkomunikasi. Tim Finin [Finin et al., 1993] [Finin et al., 1994] [Finin et al., 1995] [Finin et al., 1997] dan Yannis Labrou [Labrou et al., 1994] [Labrou et al., 1997] adalah peneliti *software agent* yang banyak berkecimpung dalam riset mengenai bahasa dan protokol komunikasi antar agent. Salah satu produk mereka adalah *Knowledge Query and Manipulation Language* (KQML). Dan masih terkait dengan komunikasi antar agent adalah *Knowledge Interchange Format* (KIF).

Software Agent bisa diklasifikasikan sebagai :

1. Desktop Agent

Yaitu agent yang hidup dan bertugas dalam lingkungan *Personal Computer* (PC), dan berjalan diatas suatu *Operating System* (OS). Yang termasuk dalam klasifikasi ini adalah:

- *Operating System Agent*
- *Application Agent*
- *Application Suite Agent*

2. Internet Agent

Yaitu agent yang hidup dan bertugas dalam lingkungan jaringan Internet, melakukan tugasnya yaitu memanage informasi yang ada di Internet. Yang termasuk dalam klasifikasi ini adalah :

- *Web Search Agent*
- *Web Server Agent*
- *Information Filtering Agent*
- *Information Retrieval Agent*
- *Notification Agent*
- *Service Agent*
- *Mobile Agent*

Bahasa Pemrograman

Bahasa pemrograman yang dipakai untuk tahap implementasi dari software agent, sangat menentukan keberhasilan dalam implementasi agent sesuai dengan yang diharapkan. Beberapa peneliti memberikan petunjuk tentang bagaimana karakteristik bahasa pemrorgaman yang sebaiknya di pakai [Knabe, 1995] [Brenner et al., 1998]. Diantaranya yaitu :

1. Object-Orientedness:

Karena agent adalah berhubungan dengan obyek, bahkan beberapa peneliti menganggap agent adalah obyek yang aktif, maka juga agent harus diimplementasikan kedalam pemrorgaman yang berorientasi obyek (object-oriented programming language).

2. Platform Independence:

Seperti sudah dibahas pada bagian sebelumnya, bahwa agent hidup dan berjalan diberbagai lingkungan. Sehingga idealnya bahasa pemrograman yang dipakai untuk implementasi adalah yang terlepas dari platform, atau dengan kata lain program tersebut harus bisa dijalankan di platform apapun (platform independence).

3. Communication Capability:

Pada saat berinteraksi dengan agent lain dalam suatu lingkungan jaringan (network environment), diperlukan kemampuan untuk melakukan komunikasi secara fisik. Sehingga diperlukan bahasa pemrograman yang dapat mensupport pemrograman yang berbasis network dan komunikasi.

4. Security:

Faktor keamanan (security) adalah factor yang sangat penting dalam memilih bahasa pemrorgaman untuk implementasi software agent. Terutama untuk mobil agent, diperlukan bahasa pemrograman yang mensupport level-level keamanan yang bisa membuat agent bergerak dengan aman.

5. Code Manipulation:

Beberapa aplikasi software agent memerlukan manipulasi kode program secara runtime, sehingga diperlikan bahasa pemrograman untuk software agent yang dapat menangani masalah runtime tersebut.

Dari karakteristik diatas dapat disimpulkan bahwa bahasa pemrograman yang layak untuk mengimplementasikan software agent adalah sebagai berikut :

- Java
- Telescript
- Tcl/Tk, Safe-Tcl, Agent-Tcl

Riset dan Aplikasi Software Agent

Ada dua tujuan dari survey tentang riset dan aplikasi software agent. Yang pertama adalah, untuk mengeidentifikasi sampai sejauh mana teknologi agent sudah diaplikasikan dengan memberikan pointer berupa contoh-contoh aplikasi sistem yang sudah ada. Yang kedua adalah, untuk memberikan gambaran ke depan, masalah-masalah apa yang sudah dan belum terpecahkan dan membuka peluang untuk mencoba mengaplikasikan teknologi agent ke masalah baru yang timbul.. Dibawah ini beberapa contoh riset Software Agent dalam bidang industri, internet/bisnis, entertainment, medis, dan bidang pendidikan.

1. Riset dan Aplikasi Software Agent di Dunia Industri

Dewasa ini teknologi agent sudah diaplikasikan secara luas di dunia Industri. dan harus diakui bahwa sejarah penelitian software agent telah berkembang. Tidak hanya dunia Internet dan bisnis, dewasa ini teknologi agent banyak didesain untuk dimanfaatkan di bidang industri.

• **Manufacturing:**

Parunak [Parunak, 1987] mempelopori proyek penelitian yang dia sebut YAMS (Yet Another Manufacturing System), dimana dia berusaha mengaplikasikan protokol contract net untuk proses kontrol di manufacturing. Untuk mengatasi masalah kompleks dalam proses manufacturing, YAMS mengadopsi pendekatan MAS, dimana setiap pabrik dan komponen didalamnya direpresentasikan sebagai agent. Aplikasi lain yang menggunakan teknologi agent dalam area ini adalah: konfigurasi dan desain untuk product manufacturing [Darrand et al., 1996], pendesainan secara kolaborativ [Cutosky et al., 1994] [Brooks, 1986], pengontrolan dan penjadwalan operasi manufacturing [Fordyce et al., 1994] [Oliveira et al., 1997] [Parunak et al., , 1997] [Sprumont et al., 1997], dsb.

• **Process Control:**

Process control secara sistem merupakan sistem yang harus bisa bekerja secara mandiri dan bersifat reactive. Hal ini sesuai dengan karakteristik dari agent, sehingga bukan sesuatu yang mengejutkan kalau banyak muncul pengembangan aplikasi process control yang berbasis ke teknologi agent. Beberapa contoh penelitian dan aplikasi yang berada dalam area ini adalah: proyek ARCHON yang diaplikasikan untuk manajemen transportasi listrik [Corera et al., 1996] dan kontrol untuk percepatan partikel [Perriolat et al., 1996], kemudian juga: pengontrolan iklim [Clearwater et al., 1996], pengontrolan spacecraft [Ingrand et al., 1992] [Schwuttke et al., 1993], dsb.

• **Telecommunications:**

Sistem telekomunikasi pada umumnya bergerak dalam skala besar, dan komponen-komponen telekomunikasi yang terhubung, terdistribusi dalam jaringan. Untuk itu diperlukan sistem monitoring dan manajemen dalam kerangka real-time. Dengan semakin tingginya tingkat kompetisi untuk menyediakan sistem komunikasi yang terbaik, diperlukan pendekatan komputerisasi dan software paradigma yang sesuai. Disinilah teknologi agent diperlukan. Beberapa riset dan aplikasi dalam area ini adalah: pengontrolan jaringan [Schoonderwoerd et al., 1997] [Weihmayer et al., 1998], transmisi dan switching [Nishibe et al., 1993], service management [Burmeister et al., 1997], dan manajemen jaringan [Esfandani et al., 1996] [Garijo et al., 1992] [Rao et al., 1990], dsb.

• **Air Traffic Control:**

Ljunberg [Ljunberg et al., 1992] mengemukakan sistem pengontrolan lalu lintas udara berbasis agent yang terkenal dengan nama OASIS. OASIS sudah diujicoba di bandar udara Sydney di Australia. OASIS diimplemantasikan menggunakan sistem yang disebut DMARS [Georgeff, 1994].

• **Transportation System:**

Beberapa contoh aplikasi teknologi agent yang ada dalam area ini adalah: aplikasi pencarian sistem transportasi dan pemesanan tiket dengan menggunakan MAS [Burmeister et al., 1997], kemudian aplikasi lain adalah seperti yang dikemukakan oleh Fischer [Fischer et al., 1996].

2. Riset dan Aplikasi Software Agent di Dunia Internet dan Bisnis

Seperti sudah disebutkan diatas, boleh dikatakan teknologi agent paling banyak diaplikasikan dalam dunia Internet dan bisnis. Bagaimanapun juga ini tak lepas dari perkembangan teknologi yang sangat pesat terutama teknologi jaringan computer. Sehingga perlu paradigma baru untuk menangani masalah kolaborasi, koordinasi dalam jarak yang jauh.

• **Information Management:**

Ada dua tema besar dalam manajemen informasi dan peran teknologi agent untuk mengatasi masalah information overload karena perkembangan teknologi jaringan dan Internet.

• **Information Filtering:**

Proyek MAXIMS [Maes, 1994] [Decker et al., 1997], kemudian WARREN [Takahashi et al., 1997] adalah contoh aplikasi di bidang information filter.

• Information Gathering:

Banyak sekali aplikasi yang masuk area information gathering baik gratis maupun komersil. Contohnya adalah proyek WEBMATE [Chen et al., 1998], pencarian homepage dengan softbot [Etzioni, 1996], proyek LETIZIA [Lieberman, 1995], dsb.

• **Electronic Commerce:**

Tema riset kearah desain dan implementasi untuk mengotomatisasi jual-beli, termasuk didalamnya adalah implementasi strategi dan interaksi dalam jual-beli, tawar-menawar, teknik pembayaran, dsb. [Chaves et al., 1996] merealisasikan sistem pasar elektronik dalam sistem yang disebut dengan KASBAH. Dalam sistem ini disimulasikan buyer agent dan seller agent yang melakukan transaksi jual-beli, tawar-menawar, dan masing-masing agent mempunyai strategi jual beli untuk mendapatkan yang termurah atau teruntung. Aplikasi agent lainnya adalah BargainFinder [Krulwich, 1996], JANGO [Doorenbos et al., 1997], MAGMA [Tsvetovatyy et al., 1997], dsb.

• **Distributed Project Management:**

Untuk meningkatkan produktivitas dalam kerja yang memerlukan kolaborasi antar anggota tim dalam kerangka teamwork, mau tidak mau harus dipikirkan kembali model software yang

mempunyai karakteristik bisa melakukan kolaborasi dan koordinasi secara mandiri, untuk membantu tiap anggota dalam melakukan tugas yang menjadi tanggung jawabnya. Salah satu pemecahannya adalah dengan mengimplemantasikan teknologi agent dalam software sistem yang dipakai untuk berkolaborasi. Anumba [Anumba et al., 1997] memberikan kontribusi dalam pengembangan decision support system untuk designer dalam mendesain bangunan dalam kerangka teamwork. Riset dan aplikasi lain adalah RAPPID [Parsons et al., 1999], PROCESSLINK [Petrie et al., 1999], dan juga OOEXPERT [Romi et al, June 1999] [Romi et al., March 1999] [Romi et al., July 2000] [Romi, 2001] yang memberikan solusi dan metodologi dalam pemecahan masalah object model creation process dalam OOAD, dan implementasi dengan menggunakan pendekatan Multi Agent System (MAS).

3. Riset dan Aplikasi Software Agent di Dunia Entertainment

Komunitas informatika dan ilmu komputer sering tidak menjamah secara serius industri-industri yang bersifat lebih ke arah rekreasi dan kesenangan (Leisure Industri) [Jennings et al., 1998]. Misalnya adalah masalah industri game, teater dan sinema, dsb. Dengan adanya software agent, memungkinkan komunitas informatika dan komputer ikut andil dalam merealisasikan industri yang bersifat entertainment ini.

• Games:

Software agent berperan penting dalam pengembangan game modern, misalnya dengan membawa paradigma agent kedalam karakter manusia atau sesuatu dalam game tersebut sehingga lebih hidup. Beberapa riset yang sudah sampai pada tahap implementasi adalah aplikasi game yang dikembangkan oleh Grand dan Cliff [Grand et al., 1998], kemudian juga [Wavish et al., 1996], dsb.

4. Riset dan Aplikasi Software Agent di Dunia Medis

Dunia medis adalah bidang yang akhir-akhir ini sangat gencar dilakukan proses komputerisasi . Tidak ketinggalan, teknologi agent pun dicoba untuk diimplementasikan dalam rangka mencoba mengatasi masalah-masalah yang berhubungan dengan monitoring pasien [Larsson et al., 1998], manajemen kesehatan dari pasien [Huang et al., 1995], dsb.