NECTEC Technical Journal

วารสารวิชาการเนคเทค ปีที่ 1 ฉบับที่ 5 เดือน พฤศจิกายน–ธันวาคม 2542 Vol. 1, No. 5, November–December 1999 ISSN 1513-2145

Characterization of InP/InGaAs/InP Single Quantum Wells Grown by OMVPE

Jiti Nukeaw, Suwan Kusamran, Anupong Srongprapa, Sanay Akavipat, and Anuchit Jaruvanawat*

Department of Applied Physics, King Mongkut's Institute of Technology Ladkrabang, Bangkok 1O520, Thailand.

Abstract-- The optical transition energies in InP/InGaAs/InP single quantum wells (SQWs) grown by organometallic vapor phase epitaxy (OMVPE) with well thicknesses from 1 to 5 monolayers(ML) were investigated by room-temperature photoreflectance (PR) measurements. PR features due to subband transitions were clearly observed even in the SQWs with extremely thin well thicknesses. PR spectra showed e(1)-hh(l) and e(1)-lh(1) transitions in the lnGaAs wells together with the band-to-band transition in the lnP layers. Clear PR spectra indicate excellent optical quality of these OMVPE-grown structures. The transition energies were determined by fitting the PR spectra to the theoretical line-shape expression. The resultant e(1)-hh(l) transition energies were close to energy positions of emission peaks observed in photoluminescence measurements at room temperature. The e(1)-hh(1) and e(1)-lh(1) transition energies decreased with increasing well thicknesses. This behavior agreed qualitatively with the theoretical prediction, although there is a significant discrepancy in transition energies.

บทคัดย่อ--การถ่ายทอดพลังงานในบ่อควอนตัมของสารกึ่งตัวนำ InP/InGaAs/InP โครงสร้างบ่อควอนตัมเดี่ยวปลูกผลึกโดยวิธี Organometallic Vapor Phase Epitaxy (OMVPE) ถูกตรวจสอบโดยวิธีวัดโฟโตรีแฟลกแตนซ์ (Photoreflectance,PR) ที่อุณหภูมิห้อง ลักษณะสเปคตรัมของ PR ที่แสดงถึงการถ่ายทอดพลังงานในบ่อที่บางอย่างยิ่งยวดถูกสังเกตุเห็นอย่างชัดเจนซึ่งเป็นการถ่ายทอดพลังงานระหว่าง e(1)-hh(1) และ e(1)-lh(1) กับการถ่ายทอดพลังงานระหว่างแถบพลังงานต้องห้ามของ InP สเปคตรัม PR ที่สังเกตุเห็นอย่างชัดเจนบ่งชี้ถึงคุณภาพของการปลูกผลึกโดยวิธีนี้ ค่าการถ่ายทอดพลังงานได้จากการ Fitting ข้อมูลที่ได้จากการทดลองเข้ากับทฤษฎี ค่าที่ได้รับนี้ใกล้เคียงกับการวัดโดยวิธีโฟโตลูมิเนสเซนต์ (Photoluminessence,PL) การถ่ายทอดพลังงานระหว่าง e(1)-hh(1) และ e(1)-lh(1) ลดลงเมื่อบ่อศักย์กว้างขึ้น พฤติกรรมนี้เหมือนกันกับการยืนยันทางทฤษฎี แม้มีค่าที่แตกต่างกัน

*Electronic mail: jiti@physicsO2.sci.kmitl.ac.th

1. Introduction

$In_{0.53}Ga_{0.47}As$ lattice-matched to InP has emerged as a very important semiconductor material. This material is promising for ultrahigh speed devices utilizing the high electron mobility and high peak velocity which is approximately 50 percent

higher than that of GaAs at comparable impurity concentrations at 300 K[1-4]. The band gap of 0.75 eV (1.65 μm) is ideal for photodetectors in optical communications systems with the optimum wavelength range between 1.3-1.6 μm. Furthermore, semiconductor injection lasers utilizing InGaAs/InP quantum well structures allow emission wavelength to be shifted from the 1.65 μm to the 1.3-1.55 μm region by having different well thickness[5,6].

Modulation spectroscopy is an important technique for study and characterization of energy-band structures of semiconductors. Modulation techniques such as electroreflectance (ER) and photoreflectance (PR) are particularly useful since they yield spectra with sharp features at critical-point energies. The features in these spectra appear at energies corresponding to the band gap characteristic points or other peculiarities in the dielectric function. PR is of considerable interest because it is contactless, requires no special mounting of the sample, can be performed in a variety of transparent ambients, and is sensitive to surface and interface electric fields.

Reddy *et al*.[7] have reported PR results on GaAs/(Al,Ga)As multiple quantum wells of different well thicknesses. Their results concern with transitions involving the so-called "unconfined" states. They also performed PR studies on a series of InGaAs/GaAs single quantum well (SQW) already defined with different well thicknesses in a range between 8 to 12 nm.[8] Their study indicated that the conduction-band discontinuity is 0.420 eV. Yaguchi *et al*.[9] studied the band offsets at the heterointerface of $GaAs/GaAs_{1-x}P_x$ SQWs structures of different well thicknesses in a range between 5 nm to 20 nm using PR. The band offsets were found to be almost linearly dependent on the phosphorus composition in the range of $x \leq 0.23$.

In this report, we conducted room-temperature PR measurements to investigate optical transition energies in InP/InGaAs/InP SQWs with extremely thin well thicknesses of 1 to 5ML, i.e., 0.3~1.5 nm, grown by a low-pressure organometallic vapor phase epitaxy (OMVPE). The subband transitions were clearly observed even in the SQWs with ultra thin well thickness, and their energies will be discussed with theoretical calculation.

2. Sample preparation

The samples were prepared using a low-pressure OMVPE. Trimethylgallium (TMGa), trimethylindium (TMIn), AsH_3, and PH_3 were used as the source gases. A 100 nm thick InP buffer layer was grown on all semi-insulating InP substrates, followed by InGaAs well layers with varied thicknesses (L_w) from 1 ML to 5 ML, and a 2 nm thick InP cap layer. The growth temperature was 600 ^{0}C. Flow sequence of source gases and the sample structure are shown in Fig. 1.

3. Photoreflectance measurement system

In our PR measurements, a modulation light was provided by a 2 mW He-Ne (λ=632.8 nm) laser. The chopped laser light was irradiated onto the sample with a spot radius of about 1 mm. A 100 W tungsten lamp was dispersed by a 20 cm monochromator and used as a probe light. The reflected probe light from the sample was detected by a nitrogen-cooled Ge detector. The detected signal contains two parts, ac part and dc part. The ac part measured by the lock-in amplifier synchronized to the modulating frequency is related to the change in reflectivity, dR. The dc part of the detected signal is related to the reflected light, R, itself. Using a computer for data acquisition and processing, a spectrum of dR/R versus photon energy can be obtained.

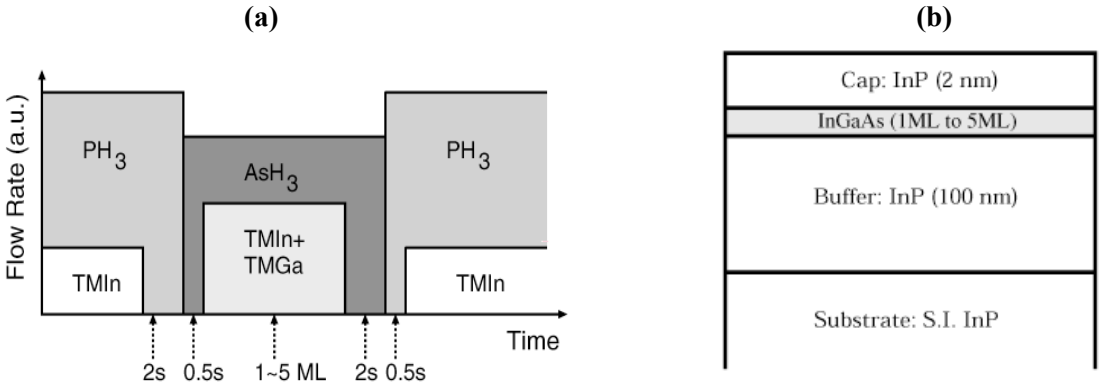


Fig. 1. (a) Flow sequence of source gases **(b)** InP/InGaAs/InP SQW structures.

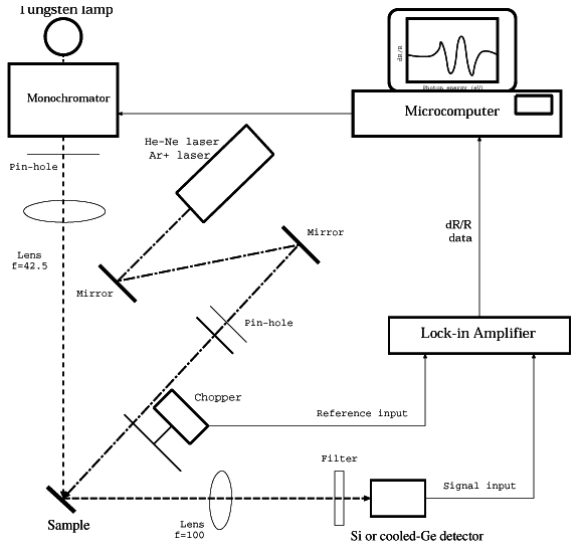


Fig. 2. Schematic diagram of the room-temperature PR measurement system.

The schematic diagram of the PR measurement system is shown in Fig. 2. All PR measurements were conducted at room temperature.

4. Results and Discussion

Fig. 3 shows PR spectrum of the sample with 2 ML (0.6 nm) thick InGaAs well. The e (1)-hh(l) and e(1)-lh(1) transitions from the InGaAs well are clearly observed with the band-to-band transition in the InP layer. For comparison, the photoluminescence (PL) spectrum measured at room temperature is also shown in Fig. 3. The resultant e(1)-hh(l) transition energy is close to the energy position of emission peak observed in the PL spectrum. PR spectra of all the samples are shown by solid lines in Fig. 4. The PR spectra reveal e(1)-hh(1) and e(1)-lh(l) transitions in InGaAs wells together with the band-to-band transition in the InP layers. The clear PR spectra indicate excellent optical quality of these OMVPE-grown structures.

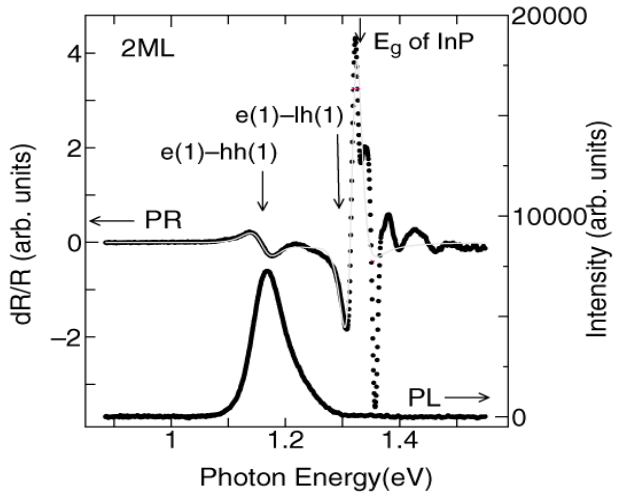


Fig. 3. Room-temperature PR and PL spectra of the InP/InGaAs/InP SQW with 2 ML (0.6 nm) thick InGaAs well.

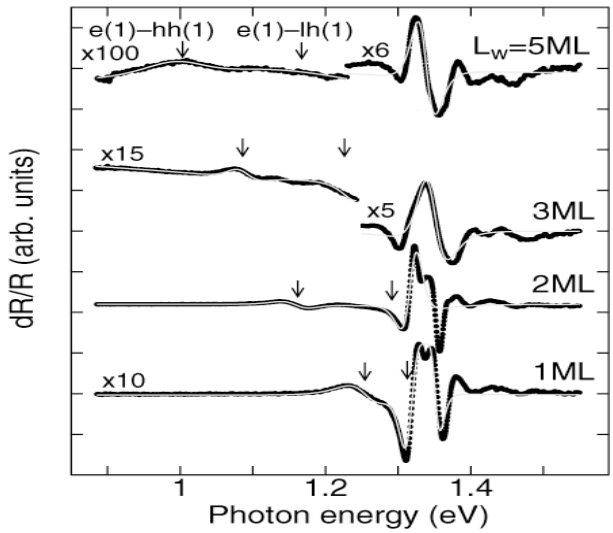


Fig. 4. Room-temperature PR spectra as a function of the well thickness. The fittings of Eq. (1) to the PR spectra are shown by bright lines. The e(1)-hh(1) and e(1)-lh(1) transition energies determined by the fittings are indicated by arrows.

The PR spectra as a function of photon energy can be analyzed using the familiar Aspnes third-derivative function in the low electric field limit'[10], i.e.,

$$\frac{\Delta R}{R} = \mathrm{Re}\sum_{j=1}^{p} C_j e^{i\theta_j} (E - E_{gi} + i\Gamma_j)^{-n} \qquad (1)$$

Here, R is the reflectance, ΔR is the induced change in the reflectance by modulation light, E is the photon energy, p is the total number of spectral structures to be fitted, E_{gi}, Γ_j, C_j and θ_j are transition energy, broadening parameter, amplitude and phase, respectively, of the feature corresponding to the j^{th} critical point. The parameter n is a factor used to specify the critical point dimension.

The energy levels associated with e(1)-hh(1) and e(l)-lh(1) transitions were determined by least-square fitting of Eq. (1) to PR spectra obtained experimentally. In this calculation, the n value for the e(1)-hh(l) and e(1)-lh(1) features is 3, while the value is 5/2 for the band gap transition[11,12]. The resultant least-square fittings are also shown by bright lines in Fig. 4. The e(1)-hh(l) and e(1)-lh(1) transition energies obtained from these fittings are shown by arrows in the figure. In Fig. 5. The e(l)-hh(l) transition energies determined from PR measurements are plotted by closed lozenges as a function of the well thickness, while open circles denote energy positions of emission peaks observed in PL measurements. The transition energy decreases with increasing well thickness. The e(l)-hh(1) transition energies obtained from PR measurements agree quite well with PL results for all samples. Figure 6 shows the well thickness dependence of the e(l)-lh(l) transition energies plotted by closed lozenges. The transition energy also decreases with increasing well thickness, which is similar to the behavior of the e(1)-hh(l) transition.

We have made a simple calculation for quantum levels in finite quantum wells. The one-dimensional Schrodinger equation for a finite square well was solved to obtain a stationary wave function. The e(1)-hh(1) and e(1)-lh(1) energies were calculated without considering any other effects such as excitons and the Stark effect. In the calculation, 0.73 eV and 1.35 eV were used as room temperature band gaps for InGaAs and InP, respectively. The conduction-band discontinuity

ΔE_c of 0.217 eV and valence-band discontinuity ΔE_v of 0.403 eV were applied[13]. The calculated transition energies are also plotted as a function of the well thickness by closed circles for the e (1)-hh(1) transition in Fig. 5 and for the e(1)-lh (1) transition in Fig. 6. The behavior of measured transition energies agrees qualitatively with the theoretical prediction, which decreases with the

increasing well thickness. However, we observed a significant discrepancy between the measured and the calculated values. The measured energies are, in general, 50 meV below the calculated energies.

Now we should discuss the origin of the discrepancy in the transition energies determined experimentally and calculated theoretically. The first possible candidate for the origin is the limit of our simple theory for the prediction of the transition energies. Yamada *et al.,* theoretically studied subband structure of the InP/InGaAs/InP SQWs using the envelope function approximation method taking into account band nonparabolicity.[14] According to their calculation, nonparabolicity plays an important role in the InP/InGaAs/InP SQWs and experimental PL energies for wide quantum wells are reasonably explained by this theory. For the well thickness less than 5 nm, the PL energies are significantly lower than the theoretical results. Gradually increasing discrepancies with decreasing well width is contradictory to the participation of impurities in the recombination process. The possibility of the participation of free excitons is, however, not excluded, since the half-widths of PL spectra are relatively broad. But the discrepancies seem to be somewhat larger than the binding energies of the exciton confined in the InGaAs wells.

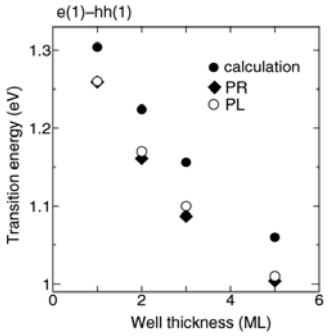


Fig. 5. Dependence of transition energy e(1)-hh(1) on well thicknesses. Closed lozenges and open circles show the PR and PL results, respectively. Closed circles show the calculated values.

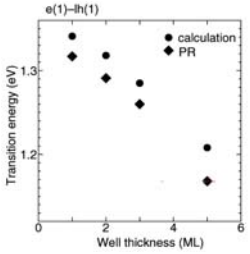


Fig. 6. Dependence of transition energy e(1)-lh(1) on well thicknesses. Closed lozenges show the PR results. Closed circles show the calculated values.

The second possible candidate for the discrepancy is imperfection of the interfaces between InGaAs and InP. The dependence of optical properties of thin InP/InGaAs/InP SQWs on the gas switching procedure during the OMVPE growth was reported by Bohrer *et al*[15]. In their study, they observed $InAs_{1-x}P_x$, islands at InGaAs/InP interfaces, which were formed as a consequence of the As-P exchange reaction. It is important to note that the islands exist at the interfaces even in the sample prepared with careful control of the gas switching procedure, although the phosphorous composition x of the $InAs_{1-x}P_x$ islands is influenced by the interruption time and the exchange efficiency of group-V elements depending on growth temperature. Based on their observation, existence of the imperfect InGaAs/InP interfaces is reasonably assumed in our samples, resulting in deviation from the ideal square shape of the potential well. The deviation induces changes in well thickness, effective ΔE_c and ΔE_v and effective bulk band gap of the well material. The resultant transition energies are obviously determined by complicated combination of these factors. Furthermore, the influence of the deviation works more significantly in the samples with the thinner well, because the fraction of the interface region is larger in total volume of the well. X-ray crystal truncation rod (CTR) scattering measurements are now in progress on the samples to reveal the atomic-level InGaAs/InP heterointerface structures. Comprehensive discussion on a basis of the atomic-level structures will be described elsewhere.

5. Conclusions

PR measurements were used to investigate optical transition energies in InP/InGaAs/InP SQWs with well thicknesses from 1 to 5 ML (from 0.3 to 1.5 nm) grown by a low-pressure OMVPE. The PR spectra clearly showed the e(l)-hh(1) and e(1)-lh(1) transitions in the SQWs together with the band-to-band transition in the InP layers. Clear PR spectra indicate excellent optical quality of these structures. The transition energies were determined by fitting the PR spectra to the theoretical line-shape expression. The resultant e(l)-hh(1) transition energies were close to energy positions of emission peaks observed in PL measurements at room temperature. The e(1)-hh(1) and e(1)-lh(l) transition energies decreased with the increasing well thickness. This behavior agreed qualitatively with the theoretical prediction, although there is a significant discrepancy in transition energies.

Acknowledgments

We would like to record our sincere appreciation to Toray Thailand Science Foundation (TTSF) for the research grant for this study, Professor Yoshikazu Takeda of Nagoya University for his gifts of samples, advice and encouragement, and King Mongkut's Institute of Technology Ladkrabang for the research facilities.

References

[1] Y. Takeda, A. Sasaki, Y. Imamura, and T. Takagi, *J. Appl. Phys*. **47** (1976) 5405.

[2] J. D. Oliver, Jr. and L. F. Eastman, *J. Electron. Mater*. **9** (1980) 693.

[3] A. Sasaki, Y. Takeda, N. Shikagawa, and T. Takagi, *Jpn. J. Appl. Phys., Supplement*, **16-1** (1977) 239.

[4] H. Ohno, and J. Barnard, "*GaInAsP Alloy Conductors*", edited by T. P. Pearsall (Wiley, New York, 1982), p. 437.

[5] W. T. Tsang*, Appl. Phys. Lett*. **44** (1984) 288.

[6] H. Temkin, K. Alavi, W. R. Wagner, T. P. Pearsall, and A. Y. Cho, *Appl. Phys. Lett*. **42**(1983) 845.

[7] U.K**.** Reddy, G. Ji, T. Henderson, H. Morkoc, and J. N. Schulman, *J. Appl. Phys*. **62** (1987) 145.

[8] U. K. Reddy, G. Ji, T. Henderson, D. Huang, and R. Houdre, and H. Morkoc, *J. Vac. Sci. Technol*. **BT** (1989) 1106.

[9] H. Yaguchi, X. Zhang, K. Ota, M. Nagahara, K. Onabe, Y. Shiraki, and R. Ito, *Jpn. J. Appl. Phys*. **32** (1993) 544.

[10] D. E. Aspnes, *Surf. Sci*. **37** (1973) 418.

[11] Y. Misiewicz, P. Markiewicz, J. Rebisz, Z. Gumienmy, M. Panek, B. Sciana, and M. Tlaczala, *Phys. Stat. Sol. (b)* **183** (1994) 143.

[12] P. Basmaji, A. M. Ceschin, M. Siu Li, O. Hipolito, A. A. Bernussi, F. Iikawa, and P. Motisuke, *Surf. Sci*. **228** (1990) 356.

[13] M. Nakao, S. Yoshida, and S. Gonda, *Solid State Commun*. **49** (1984) 663.

[14] S. Yamada, A. Taguchi, and A.Sugimura, *Appl.Phys.Lett*.**46** (1985) 675.

[15] J. Bohrer, A. Krost, and D. Bimberg, *Appl. Phys. Lett*. **60** (1992) 2258.

Jiti Nukeaw

He received B.Ed(Physics) from Srinakarinwirot University, Songkla in 1983 and M.S. (Physics) from Chiangmai University in 1989 and D.Eng.(Material Sciences and Engineering) from Nagoya University in 1998. At present, his working as lecture at Department of Applied Physics, KMITL. His current research interests include Quantum Well Device, Semiconductor Physics and Optical Instrumentation.

Reducing Data Hazards on Multi-pipelined DSP Architecture with Loop Scheduling*

Sissades Tongsima
NECTEC/HPC
Bkk, Thailand

Chantana Chantrapornchai
Silpakorn University
Bkk, Thailand

Edwin H.-M. Sha
U. of Notre Dame
Indiana, USA.

Nelson L. Passos
Midwestern St.
Texas, USA.

Abstract—Computation intensive DSP applications usually require parallel/pipelined processors in order to meet specific timing requirements. Data hazards are a major obstacle against the high performance of pipelined systems. This paper presents a novel efficient loop scheduling algorithm that reduces data hazards for such DSP applications. This algorithm has been embedded in a tool, called SHARP, which schedules a pipelined data flow graph to multiple pipelined units while hiding the underlying data hazards and minimizing the execution time. This paper reports significant improvement for some well-known benchmarks showing the efficiency of the scheduling algorithm and the flexibility of the simulation tool.

บทคัดย่อ—งานประยุกต์ทางด้านดิจิตอลซิกแนลโปรเซสซิ่งที่ต้องการความสามารถในการคำนวณสูงมักจะต้องการหน่วยประมวลผลแบบขนานที่มีการซ้อนลำดับของการคำนวณ ของโปรแกรม(parallel/pipelined processors) เข้ามาช่วยในการคำนวณเพื่อที่จะ ทำให้ได้ผลลัพท์ทันเวลาที่ต้อง การ แต่อย่างไรก็ตามหน่วยประมวลผลแบบขนานเหล่านี้มักจะมีจุดอ่อนอยู่ที่ความจำเป็นที่ต้องรอข้อมูลซึ่งยังไม่ถูกคำนวณ (data hazards) ที่ทำให้การซ้อนลำดับของการคำนวณของโปรแกรมไม่ได้ช่วยการคำนวณให้เร็วขึ้น บทความนี้นำเสนอวิธีการช่วย ลดความจำเป็นที่ต้องรอข้อมูลซึ่งยังไม่ถูกคำนวณในงานประยุกต์ทาง ด้านดิจิตอลซิกแนลโปรเซสซิ่งนี้โดยนำเอาหลักการที่ชื่อว่าลูปไพพ์ไลน์นิ่งเข้ามาช่วย

1 Introduction

In order to speedup current high performance DSP systems, *multiple pipelining* is an important strategy that should be explored. Nonetheless, it is well-known that one of the major problems in applying the pipelining technique is the delay caused by dependencies between instructions, called *hazards*. Control hazards are known as the hazards that prevent the next instruction in the instruction stream from being executed, such as branch operations. Likewise, the hazards that encumber the next instruction by data dependencies are called data hazards. Most computation-intensive scientific applications, such as image processing, and digital signal processing, contain a great number of *data hazards* and few or no *control hazards*. In this paper, we present a tool, called *SHARP* (Scheduling with Hazard Reduction for multiple Pipeline architecture), which was developed to obtain a short schedule while minimizing the underlying data hazards by exploring loop pipelining and different multiple pipeline architectures.

Many computer vendors utilize a *forwarding* technique to reduce the number of data hazards in their architectures. This process is implemented in hardware whereby a copy of the computed result is sent back to the input prefetch buffer of the processor. However, the larger the number of forwarding buffers, the higher the cost that will be imposed on the hardware. Therefore, there exists a trade-off between its implementation cost and the performance gain. Furthermore, many modern high speed computers, such as MIPS R8000, IBM Power2 RS/6000 and others, use multiple pipelined functional units (multi-pipelined) or superscalar (super)pipelined architectures. Providing a tool that determines an appropriate pipelined architecture for a given specific application, therefore, will be beneficial to computer architects. By using such a tool, one can find a suitable pipeline architecture that balances the hardware and performance costs by varying the system architecture (e.g., a number of pipeline units, type of each unit, forwarding buffers, etc.).

Rearranging the execution sequence of tasks that belong to the computational application can reduce data hazards and improve the performance. Dynamic scheduling

*Reprint with permission from Kluwer Academic Publishers in the journal of VLSI signal processing, special issue on future directions in the design and implementation of DSP systems, Vol 18, 1998, pp 111-123

algorithms such as *tomasulo* and *scoreboard* are examples of implementing the algorithms in hardware. They were introduced to minimize the underlying data hazards which can not be resolved by a compiler [16]. These techniques, however, increase the hardware complexity and costs. Therefore, special consideration should be given to static scheduling, especially for some computation-intensive applications. The fundamental performance measurement of a *static* scheduling algorithm is the total completion time in each iteration, also known as the schedule length. A good algorithm must be able to maximize parallelism between tasks and minimize the total completion time. Many heuristics have been proposed to deal with this problem, such as *ASAP* scheduling, *ALAP* scheduling, *critical path* scheduling and *list* scheduling algorithms [2, 3]. The critical path, list scheduling and graph decomposition heuristics have been developed for scheduling acyclic *data flow graphs (DFGs)* [7, 14]. These methods, however, do not consider the parallelism and pipelining across iterations. Some studies propose scheduling algorithms to deal with cyclic graphs [5, 15]. Nevertheless, these techniques do not address the issue of scheduling on pipelined machines that exploit the use of forwarding techniques.

Considerable research has been done in the area of loop scheduling based on *software pipelining*—a fine-grain loop scheduling optimization method [6, 10, 12]. This approach applies the *unrolling* technique which expands the target code segment. The problem size, however, also increases proportionally to the unrolling factor. *Iterative modulo scheduling* is another framework that has been implemented in some compilers [13]. Nonetheless, in order to find an optimized schedule, this approach begins with an infeasible initial schedule and has to reschedule *every* node in the graph at each iteration.

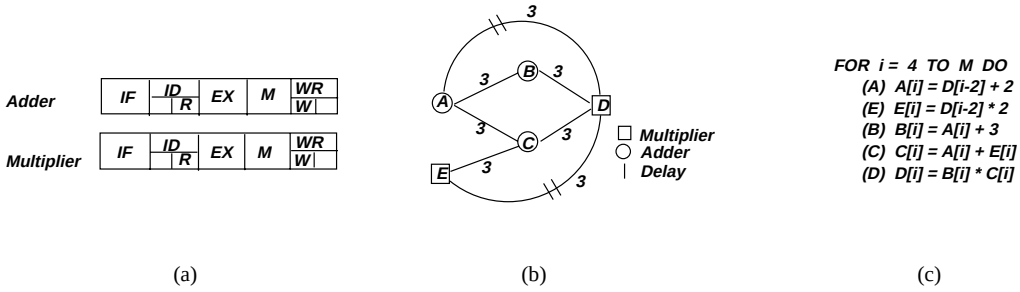
The target DSP applications usually contain iterative or recursive code segments. Such segments are represented in our new model, called a *pipeline data-flow graph* (PDG). An example of a PDG is shown in Figure 1(b). In this model, nodes represent tasks that will be issued to a certain type of pipeline and edges represent data dependencies between two nodes. A weight on each edge refers to a minimum hazard cost or pipeline cost. This cost represents a required number of clock cycles that must occur in order to schedule successive nodes. In this work, a proposed novel pipeline scheduling algorithm, SHARP, takes a PDG and some pipeline architecture specifications (e.g., pipeline depth, number of forwarding buffers, type and number of pipeline units etc.) as inputs. The algorithm then efficiently schedules nodes from the PDG to the target system.

After the initial schedule is computed, by a DAG scheduling algorithm such as list scheduling, the algorithm implicitly uses retiming. Only a *small* number of nodes are rescheduled in each iteration of our algorithm. The new scheduling position is obtained by considering data dependencies and loop carried dependencies, i.e., using loop pipelining strategy as a basis to reduce data hazards while improving the total execution time under the hardware constraints given by the user specifications.

As an example, Figure 1(a) presents two pipeline architectures each of which consists of five stages: instruction fetch (IF), instruction decode (ID), execution (EX), memory access (M) and write-back (WR). For simplicity, assume that each of these stages takes one *clock cycle* to finish [4]. The pipeline hazard in this case is 3, since with this architecture, any instruction will put data available to read in the first half of the 5th stage (WR) and read it in the 2nd stage (ID). In Section 2, we will explain how to calculate this cost in more detail. The PDG and its corresponding code segment to be executed in this two-pipeline system are shown in Figures 1(b) and (c) respectively. Notice that each node of the graph also indicates the type of instruction required to be executed. Assume that the WR and ID stages of two dependent instructions can be overlapped. For example, instruction *B* can start reading (at the ID stage) data produced by instruction *A* at the WR stage. A legal execution pattern of the pipeline for this example is illustrated in Figure 2(a).

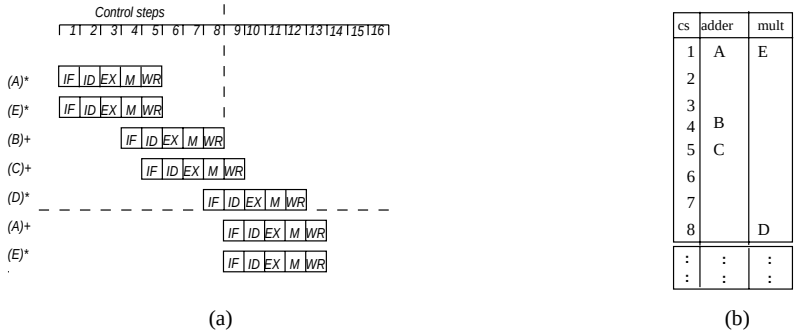
Since all the pipeline stages of issued instructions are consecutive, only the beginning of each instruction pipeline is required to be shown. Figure 2(b) illustrates a schedule table resulting from Figure 2(b). This table only shows one iteration of the sample code segment (the complete table comprises of $M-3$ identical copies of this table). Such a schedule becomes an *initial schedule* which can be optimized by SHARP. Figure 3(a) and 3(b) show the resulting intermediate PDG and schedule after applying SHARP to the initial schedule. Nodes A and E from the next iteration are rescheduled to current iteration of the schedule. This is equivalent to retiming these nodes in the PDG (see Figure 3(a)). This technique explores the parallelism across iterations (loop pipelining). SHARP repeatedly applies such a method to each intermediate schedule. Figures 3(c) and 3(d) show the third intermediate retimed PDG and its schedule respectively. At the third iteration we obtain the optimized schedule with length 6 (a 25% improvement over the initial schedule).

Using our tool, we obtain not only the reduced schedule length but we can also evaluate other architecture options, such as introducing forwarding hardware in the architecture or even additional pipelines. In order to present our algorithm, the remainder of this work is organized as follows: Section 2 introduces some fundamental concepts. The main idea and theorems behind the algorithm used in SHARP are presented in Section 3. In Section 4, we discuss the experimental results obtained by applying different pipeline architectures to this tool. Finally, Section 5



(a) (b) (c)

Figure 1: (a) Target pipeline architecture (b) Pipeline DFG (c) Corresponding code segment

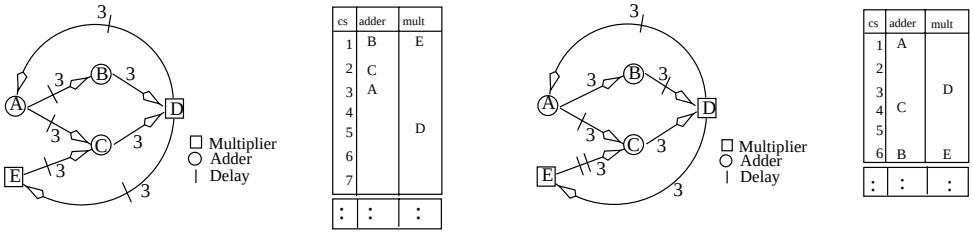


(a)

cs	adder	mult
1	A	E
2		
3		
4	B	
5	C	
6		
7		
8		D
:	:	:

(b)

Figure 2: (a) Pipeline execution pattern (b) An equivalent schedule



cs	adder	mult
1	B	E
2	C	
3	A	
4		
5		D
6		
7		
:	:	:

(a) (b)

cs	adder	mult
1	A	
2		
3		D
4	C	
5		
6	B	E
:	:	:

(c) (d)

Figure 3: (a)-(b) PDG and schedule of intermediate step of the algorithm, (c)-(d) PDG and the schedule after third step

draws conclusions of this work.

2 Background

A *cyclic* DFG, $G = \langle V, E \rangle$, is commonly used to represent dependencies between instructions in an iterative or a recursive loop. However, it does not reflect the type of pipeline architecture to which the code segment is subjected. Therefore, in order to distinguish them, some common hardware factors need to be considered.

The type of architecture may be characterized by considering different types of pipelines in the system. The number of stages in a pipeline, or *pipeline depth*, is one configuration factor that is necessary to be taken into account, since it affects the overall pipeline speedup. The forwarding hardware is also a factor because it can diminish the data hazards. Furthermore, the system may consist of a number of forwarding buffers, responsible for how many times a pipeline is able to bypass a datum [9].

In this paper, we assume our algorithm guarantees that no delays occur during the execution of one instruction. In other words, the number of cycles from the execution of the first to the last pipeline stage for one instruction is equal to the pipeline depth. In a multi-pipelined machine, if the execution of an instruction I_2 depends on the data generated by instruction I_1, and the starting moment of I_1 and I_2 are t_1 and t_2 respectively, we know that $t_2 - t_1 \geq S_{\text{out}} - S_{\text{in}} + 1$, where S_{out} is the index of the pipeline stage from which the data is visible to the instruction I_2, and S_{in} is the index of pipeline stage that needs the result of I_1 in order to execute I_2. We call $S_{\text{out}} - S_{\text{in}} + 1$ the *pipeline cost* of the edge connecting the two nodes, representing instructions I_1 and I_2. Figure 4 illustrates the concept of the pipeline cost. Such a cost can be qualified in three possible situations depending on the characteristics of the architecture:

case 1: The pipeline architecture does not have a forwarding option. The pipeline cost is similar to the data hazard, which may be calculated from the difference between the pipeline depth and the number of the overlapped pipeline stages. For example in Figure 4(a), this pipeline reads data at the end of ID and the data is ready after WR. The S_{out} stage is 7 and S_{in} is 3. Hence, for this case, the pipeline cost is $7 - 3 + 1 = 5$.

case 2: The pipeline architecture has an *internal* forwarding, i.e., data can merely be bypassed inside the same functional unit. The pipeline cost from this case may be obtained in a similar way as above. For instance, the pipeline in Figure 4(b) has the internal forwarding such that the data will be available right after the EX stage. Then, S_{out} is stage 5 and S_{in} is stage 3, so the pipeline cost is $5 - 3 + 1 = 3$. In this case, the special forwarding hardware is characterized into two sub-cases.

1. The forwarding hardware has a limited number of feedback buffers. The pipeline cost will be the value without forwarding when all the forwarding buffers are utilized.
2. The forwarding hardware has an unlimited number of feedback buffers (bounded by the pipeline depth). In this case, the pipeline cost will always be the same.

case 3: The pipeline architecture has an *external* or *cross* forwarding hardware, such as it is capable of passing data from one pipeline to another pipeline. We calculate the pipeline cost by the same way described above. Again, limited number of buffers or unlimited number of buffers are possible sub-cases.

2.1 Graph Model

In order to model the configuration of each multi-pipelined machine associated with the problem being scheduled, the pipeline data-flow graph is introduced.

Definition 2.1 *A pipeline data-flow graph (PDG)* $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$ *is an edge-weighted directed graph, where* V *is the set of nodes,* $E \in V \times V$ *is the set of dependence edges,* T *is the pipeline type associated with a node* $u \in V$, $\mathbf{d}$ *is the number of delays between two nodes, and* $\mathbf{c}(e) = (c_{fo}, c_{no})$ *is a function from* E *to the positive integers representing the pipeline cost, associated with edge* $e \in E$, *where* c_{fo} *and* c_{no} *are the cost when considering with and without forwarding capability respectively.*

Each node in a PDG represents an instruction, and the type of pipeline in which the instruction will be executed. An edge from node u to node v, exhibited by the notation $u \rightarrow v$, conveys that the instruction v depends on the result from the instruction u. The number of delays $\mathbf{d}(e)$ on any edge $e \in E$ such that u precedes v, where $u, v \in V$, indicates a data dependence from node u to v, such that the execution of node v at iteration j relies on the data produced by node u at iteration $j - \mathbf{d}(e)$. The tuple associated with each edge in a PDG, $u \xrightarrow{(c_{fo}, c_{no})} v$, is architecture-dependent where c_{fo} is the number of clock cycles required when there exists a forwarding hardware, and c_{no} is the number of clock cycles needed when executing the two instructions u and v considering no forwarding. If there is no forwarding hardware, the value of c_{fo} will be the same as c_{no}.

As an example, Figure 5(a) illustrates a simple PDG associated with two types of functional units, adder and multiplier. Each of which is a five-stage pipeline architecture with a forwarding function. Therefore, the pipeline

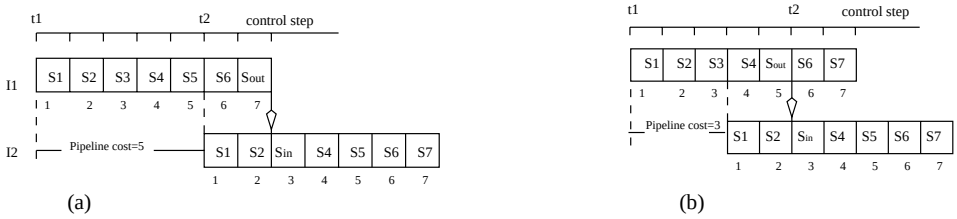


Figure 4: (a) Pipeline cost w/o forwarding hardware (b) Pipeline cost w/ forwarding capability

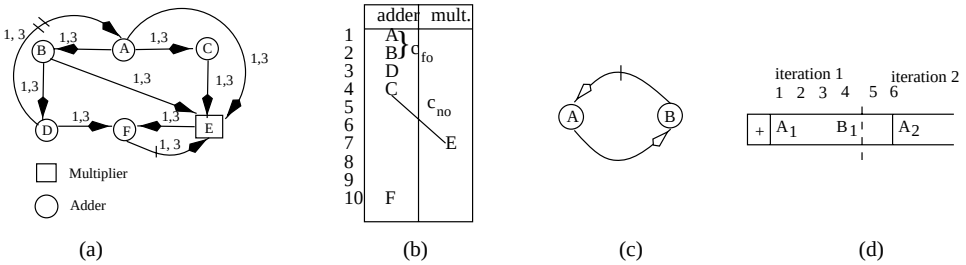


Figure 5: (a) Example when $c_{fo} = 1$ and $c_{no} = 3$ (b) Corresponding initial schedule (c)-(d) PDG with inter-iteration dependency between node *A* and *B*

cost $c_{fo} = 1$ and $c_{no} = 3$. Nodes A, B, C, D, and F represent the addition instructions and node E indicates the multiplication instruction. The bar lines on $D \rightarrow A$ and $F \rightarrow E$ represent the number of delays between the nodes, i.e., two delays on $D \rightarrow A$ conveys that the execution of operation D at some iteration j produces the data required by A at iteration $j+2$.

2.2 Initial Scheduling in SHARP

In this subsection, we introduce some important considerations in acquiring a static pipeline schedule from the PDG. Considerable research has been conducted in seeking a scheduling solution for a DFG [8]. In this research, we tailor the list scheduling heuristic so that it agrees with conditions of the PDG.

A static schedule consists of multiple entries in a table. Each row entry indicates one clock cycle—the synchronization time interval, also called *control step*. Each column entry represents one of the pipeline units in the multi-pipelined system where nodes that have the same corresponding types of pipeline will be assigned. The first pipeline stage of each scheduled node starts executing whenever the node appears in the table.

In order to obtain a static schedule from a PDG feedback edges (i.e., edges that contain delays) are temporarily ignored in this initial scheduling phase. For instance, $D \rightarrow A$ and $F \rightarrow E$ in Figure 5(a) are temporarily ignored. Our scheduling guarantees the resulting initial schedule is legal by satisfying the following properties. Further, the following scheduling properties must be preserved by any scheduling algorithm.

Property 2.1 *Scheduling properties for intra-iteration dependencies*

1. *For any node n preceded by nodes m_i by edges e_i such that $\mathbf{d}(e_i) = 0$ and $m_i \xrightarrow{(c_{fo_i}, c_{no_i})} n$. If $\mathrm{cs}(m_i)$ is the control step to which m_i was scheduled and b_i is the number of available buffers for the functional unit required by m_i, then node n can be scheduled at any control step (cs) that satisfies the following rules.*

$$cs \geq \max_i \{\mathrm{cs}(m_i) + cost(b_i)\}$$

$$where \begin{cases} cost(b_i) = c_{fo} & if\ b_i > 0 \\ cost(b_i) = c_{no} & otherwise \end{cases}$$

2. *If there is no direct-dependent edge between nodes q and r, i.e. $\mathbf{d}(e) \neq 0$, and q is scheduled to control step k, node r may be placed at any unoccupied control step which does not conflict with any other data dependency constraints.*

Note that if the architecture does not use forwarding hardware, then we set $c_{fo} = c_{no}$ and $b_i = 0$. As an example, Figure 5(b) presents the resulting schedule table when we schedule the graph shown in Figure 5(a) to a multiple pipelined system consisting of one adder and one multiplier with one internal buffer for each unit.

The last step of initial scheduling is to check the inter-iteration dependency which is implicitly represented by the

feedback edges of the PDG. A certain amount of empty control steps has to be preserved at the end of the schedule table if the number of control steps between two inter-dependent nodes belonging to different iterations is not sufficient to satisfy the required pipeline cost of the two corresponding instructions. Figures 5(c) and (d) illustrates this situation. Assume that the pipeline depth of the adder is 5. If we did not consider the feedback edge, the schedule length would be only 4. However, the schedule length actually has to be 6 since node A in the next iteration, represented by A_2, cannot be assigned to the control step right after node B_1 due to the inter-iteration dependency between nodes A and B. Hence two empty control steps need to be inserted at the end of this initial schedule and the final schedule length becomes six rather than four.

Note again that the execution of all pipeline stages, except for the first one, of any scheduled node are hidden in an initial schedule table. Those stages are overlapped and only the first stage of each node is displayed, e.g., see Figure 2(a). After applying a list scheduling algorithm that enforces Property 2.1 to the example in Figure 1, the initial schedule of Figure 2(b) is produced. The static schedule length for that case is 8.

3 Reducing Schedule Length

In the previous section, we discussed the scheduling conditions for assigning nodes from a PDG to a schedule table. These conditions are also applied to the optimization process of our algorithm. Our algorithm is able to reduce the underlying static schedule length of an initial schedule previously obtained. It explores the parallelism across iterations by implicitly employing the retiming technique [11]. The following section briefly reviews the retiming and rotation techniques.

3.1 Retiming and Rotation

The *retiming* technique is a commonly used tool for optimizing synchronous systems. A retiming $\mathbf{r}$ is a function from V to Z. The value of this function, when applied to a node v, is the number of delays taken from all incoming edges of v and moved to its outgoing edges. An illegal retiming function occurs when one of the retimed edge delays becomes negative. This situation implies a reference to a non-available data from a future iteration. Therefore, if we consider $G_{\mathbf{r}} = \langle V, E, T, \mathbf{d_r}, \mathbf{c} \rangle$ to be a PDG G retimed by a function $\mathbf{r}$, a retiming is legal if the retimed delay count $\mathbf{d_r}$ is nonnegative for every edge in E. For an edge $u \rightarrow v$, the number of additional delays is equal to the number of delays moved to the edge through node u, subtracted by the number of delays drawn out from the edge through node v. The retiming technique can be summarized by the following properties:

Property 3.1 *Let $G_{\mathbf{r}} = \langle V, E, T, \mathbf{d_r}, \mathbf{c} \rangle$ be a PDG $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$ retimed by $\mathbf{r}$.*

1. *$\mathbf{r}$ is a legal retiming if $\mathbf{d_r}(e) \geq 0$ for every $e \in E$.*
2. *For any edge $u \xrightarrow{e} v$, we have $\mathbf{d_r}(e) = \mathbf{d}(e) + \mathbf{r}(u) - \mathbf{r}(v)$.*
3. *For any path $u \overset{p}{\rightsquigarrow} v$, we have $\mathbf{d_r}(p) = \mathbf{d}(p) + \mathbf{r}(u) - \mathbf{r}(v)$.*
4. *For a loop l, we have $\mathbf{d_r}(l) = \mathbf{d}(l)$.*

Property 3.1 demonstrates how the retiming method operates on a PDG. An example of retiming is shown in Figure 6. The retiming $\mathbf{r}(A) = 1$ conveys that one delay is drawn from the incoming edge of node A and pushed to all of its outgoing edges, $A \rightarrow B$ and $A \rightarrow C$.

After a graph has been retimed, a *prologue* is the set of instructions that must be executed to provide the necessary data for the iterative process. In our example, the instruction A becomes the prologue. An *epilogue* is the other extreme, where a complementary set of instructions will need to be executed to complete the process. The time required to run the prologue and epilogue is assumed to be negligible when compared to the total computation time of the problem.

Chao, LaPaugh and Sha proposed a flexible algorithm, called rotation scheduling, to deal with scheduling a DFG under resource constraints [1]. Like its name, this algorithm analogously moves nodes from the top of a schedule table to its bottom. The algorithm essentially shifts the iteration boundary of the static schedule down, so that nodes from the next iteration can be explored. We now introduce some necessary terminology and concepts used in this paper.

Definition 3.1 *Given a PDG $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$ and $R \subset V$, the rotation of R moves one delay from every incoming edge to all outgoing edges of nodes in R. The PDG now is transformed into a new graph (G_R).*

For a schedule table with length L, this definition is applicable when moving the first row of the schedule table to the position $L+1$. Therefore, this operation implicitly retimes the graph. The benefit of doing the rotation is that a few number of nodes are rescheduled. Therefore only a small part of an input graph is modified instead of rescheduling the whole graph. Rotation scheduling must preserve the following property:

Property 3.2 *Let $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$ be a PDG and $R \subset V$. A set R can be legally retimed if and only if every edge from $V - R$ to R contains at least one delay.*

Figure 6: An example of retiming data-flow graph (a) Original (b) When $\mathbf{r}(A) = 1$

This property implies that the rotation operation always preserves Property 3.1. After performing the rotation strategy, the dependencies in a new graph are changed, since some delays in the graph are now moved to new edges. This allows us to explore the possibility of parallelizing those nodes that do not have a direct-dependent edge from their predecessors. Carefully re-scheduling those rotated nodes to new positions, the schedule length can be decreased.

Nevertheless, as mentioned earlier, we also have to consider the inter-iteration dependency. Hence a new schedule position assignment for a node has to be carefully chosen to avoid conflicts in the dependency constraint between iterations. Finding a valid scheduling position now becomes more complex since the problem now has incorporated pipeline hazards. The major different from the traditional algorithm is that our algorithm requires checking not only dependencies of the new graph after rotating a node but also pipeline hazards which may occur only if schedule a node to different processors. The following section discusses how to find such a dependency and avoiding the underlying pipeline hazards in detail.

3.2 Minimum Schedule Length

We know that a number of delays on any edge $u \rightarrow v$, where $u, v \in V$, indicates in which iteration, prior to the current iteration, node u legally produces data for node v. This conveys that in order to schedule the rotated nodes, the pipeline cost constraints must also be satisfied, e.g., the inter-iteration dependency between nodes u and v.

Definition 3.2 *Given a PDG $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$ and nodes $u, v \in V$ where $u \rightarrow v \in E$, the minimum schedule length with respect to nodes u and v, $ML(u,v)$, is the minimum schedule length required to comply with all data-dependent constraints.*

The following theorem presents the *ML* function.

Theorem 1 *Given a PDG $G = \langle V, E, T, \mathbf{d}, \mathbf{c} \rangle$, an edge $e = u \rightarrow v \in E$, and $\mathbf{d}(e) = k$ for $k > 0$, a legal schedule length for G must be greater than or equal to $ML(u,v)$, where*

$$ML(u,v) = \left\lceil \frac{pipe\_cost + cs(u) - cs(v)}{k} \right\rceil$$

with $cs(node)$ being the starting control step of that node and pipe_cost is either c_{no} or c_{fo} depending on the architecture.

Proof: Let L be the schedule length for one iteration. We know that the minimum number of control steps between node u at iteration j and node v at iteration $j+k$ is the pipeline cost associated with $u \rightarrow v$. There are $k-1$ iterations between iterations j and $j+k$. Since all iterations have the same length L, the following equation is the relationship of the distance between $cs(u)$ and $cs(v)$: $L \times (k-1) + (L - cs(u)) + cs(v) + \Delta \geq pipe\_cost$ where Δ represents a number of compensated control steps fulfilling the pipeline cost requirement. Hence, Δ can be expressed as: $\Delta \geq pipe\_cost - L \times (k-1) - L + cs(u) - cs(v)$. In order to obtain a uniform schedule, Δ is distributed over all k iterations preceding iteration $i+k$. This distribution results in a minimum value $\delta = \lceil \frac{\Delta}{k} \rceil$, and the new static schedule length that satisfies the constraints with respect to u is $ML = \delta + L$. After substituting, we obtain

$$ML(u,v) = \left\lceil \frac{pipe\_cost + cs(u) - cs(v)}{k} \right\rceil$$

Since a node may have more than one predecessor, in order to have a legal schedule length, one must consider the maximum value of *ML*. In other words, the longest schedule length that is produced by computing this function will be the worst case that can satisfy all predecessors.

3.3 Algorithm

The scheduling algorithm used in SHARP applies the *ML* function to check if a node can legally be scheduled at a specific position. Therefore, it may happen that the obtained schedule will require some empty slots to be added to compensate for the inter-iteration dependency situation. We summarize this algorithm in Figures 7 which demonstrates how we utilize the two important optimization functions *Pipe_rotate* and *Re-schedule* in SHARP. Note again

INPUT : $\mathsf{G} = (\mathsf{V}, \mathsf{E}, \mathsf{T}, \mathbf{d}, \mathbf{c})$, # forwarding buffers, and # pipelines
OUTPUT : shortest schedule table S
begin
  $S :=$ *Start-Up-Schedule*(G); $Q := S$;
  for $i := 1$ **to** $|\mathsf{V}|$ **step** 1 **do**
    $(\mathsf{G}, S) :=$ *Pipe_rotate*(G);
    if *length*$(S) <$ *length*(Q) **then** $Q := S$; **fi** **od**
  proc *Pipe_rotate*(G, S) $\equiv$
    $\lceil N :=$ *Deallocate*(S); /* extract nodes from the table */
    $\mathsf{G}_r :=$ *Retime*(G, N); /* operate retiming technique on the nodes */
    $S :=$ *Re-schedule*(G, S, N);
    return$(\mathsf{G}_r, S) \rfloor$.
  proc *Re-schedule*(G, S, N) $\equiv$
    $\lceil$**foreach** $v \in N$ **do**
      $cs\_min := \max\{parents(v).cs + cost(parent(v).b_i)\}$;
      $cs\_max :=$ *length*(S); /* get an upper bound schedule length */
      $cs := cs\_min$;
      /* find a minimum available legal control step to schedule v */
      while $(cs < cs\_max) \wedge ((legal(cs, v, \mathsf{G}) =$
          false$) \vee ((pid := entry(S, cs)) =$ **not available**$))$ **do**
          $cs++$; **od**
      if $cs \geq cs\_max$
        then *assign*$(S, v, cs\_max, v.pid)$; /* assign at the same *pid* */
        else *assign*(S, v, cs, pid); **fi** /* assign at the obtained *pid*/
    od
    return$(S) \rfloor$.
end

Figure 7: The SHARP framework: showing how the optimization functions play their roles

that the initial schedule in the algorithm can be obtained by any DAG scheduling algorithm, e.g., a modified list scheduling that satisfies Property 2.1. Next, the procedure *Pipe_rotate* is applied to shorten the initial schedule table. It first deallocates nodes from the schedule table. These nodes are then retimed and consequently rescheduled. The procedure *Re-schedule* finds a proper position such that the schedule length will not exceed the previous length. A scheduling position has to satisfy Property 2.1 and Theorem 1. This process is computed in the while loop (Lines 8–10) which calculates an appropriate control step considering a pipeline cost and buffer size as well as *ML*. Then, if the obtained control step is smaller than the current one and a required unit is available, the node can be re-scheduled. Otherwise, it remains at the same position. As a result, the new schedule table can either be shorter or have the same length.

Consider now the PDG shown in Figure 8. In this example, there are 5 addition-instructions and 1 multiplication-instruction. Assume that the target architecture is similar to the one presented in the introduction section (i.e., one adder and one multiplier with one-buffer internal forwarding). After obtaining the initial schedule, shown in Figure 8(b), the algorithm attempts to reduce the schedule length by calling the function *Pipe_rotate* which brings A from the next iteration, called A_1, and re-schedule it to $cs5$ (which is $cs4$ after re-numbering the table) of the addition unit. By doing so, the forward buffer of A, which was granted to B in the initial schedule, is free since this new A_1 does not produce any data for B. Then, the static schedule length becomes 9 control steps. After running SHARP for 4 iterations, the schedule length is reduced to six control steps as illustrated in Figure 8(c).

4 Experimental Results

We have used SHARP in experiment on several benchmarks with different hardware assumptions: no forwarding, one buffer-internal forwarding, sufficient buffer-internal forwarding (in-frw.), one buffer-external forwarding, two buffer-external forwarding and sufficient buffer-external forwarding (ex-frw.). The target architecture is comprised of a 5-stage adder and a 6-stage multiplier pipeline units. When the forwarding feature exists, the data produced at the end of the EX-stage can be forwarded to the next execution cycle of EX-stage as shown in Figure 4(a).

Note that the *sufficient*-xx forwarding assumption conveys that its architecture has sufficient number of forwarding buffers. Furthermore, the internal and external modifiers for each assumption convey that the forwarding technique can be done within one functional unit and between two functional units respectively. The set of benchmark problems and their characteristics are shown in Figure 4(b) Tables 1 and 2 exhibits the simulation results from a system that contains one adder/one multiplier and 2 adders/2 multipliers respectively. Note that the results presented in these tables were collected after running SHARP against each benchmark until there is no improvements for 7 consecutive intermediate schedules (i.e., seven iterations of the algorithm). Both tables present an initial schedule length of each benchmark and the final length after applying the algorithm to the initial schedule (see column int/aft). The reduction percentage of each benchmark is presented in column %.

From experiments, the performance of the one buffer-internal forwarding scheme is very close to the sufficient buffer-internal forwarding one. This is because most of the selected benchmarks have only one or two outgoing edge(s) (fan-out degree) for each node. Increasing the number of internal forwarding buffers may slightly increase performance. The performance of a system with one buffer could be worse than the one with sufficient buffers for some applications with large fan-out degrees. In this case, only one successive node can be scheduled earlier by consuming the data from an only buffer and the rest of the successive nodes would cause the underlying data-dependent hazards, i.e., waiting for data being ready from its parent at WR-stage. For a system with external forwarding, data can be forwarded to any functional unit in the system. Therefore, the resulting schedule length is shorter than that of the system with internal forwarding capability.

Selecting an appropriate number of buffers depends on the maximum fan-out degree and the pipeline depth. In some cases only one or two buffers are enough with additional buffers not producing a significant improvement. As an example, consider column 4 of Table 1 which describes a system with external forwarding. Particularly for the wave digital filter application (benchmark 4) using only one buffer is the most appropriate since the algorithm results in the maximum reduction, 33%, over the initial schedule length. Adding 2 or more buffers results in an 11% reduction. For the differential equation solver application (benchmark 2), selecting two buffers is a good choice since the algorithm yields the maximum reduction.

The number of available units is also another signifi-

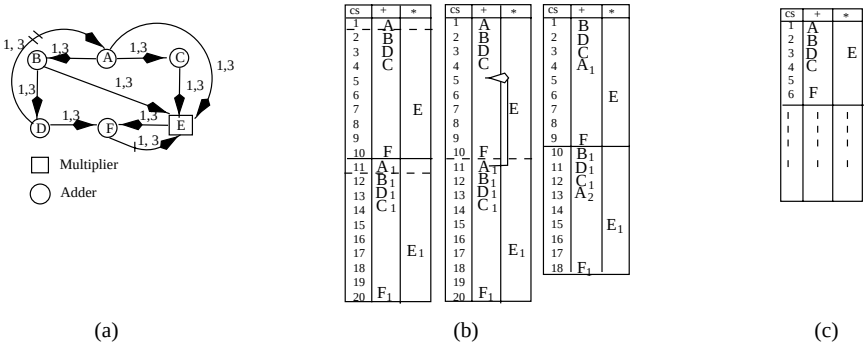
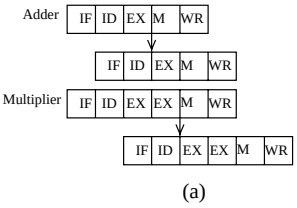


Figure 8: (a) The 6-node PDG (b) the first optimized schedule table and (c) the final schedule



Benchmark	Mul	Add	Sum
1. 5-th Order Elliptic Filter	8	26	34
2. Differential Equation Solver	6	5	11
3. All-pole Lattice Filter	11	4	15
4. WDF for Transmission Line Computation	4	8	12
5. Unfolded All-pole Lattice Filter (uf=7)	66	24	90
6. Unfolded Differential Equa. Solver (uf=5)	24	20	44
7. Unfolded 5-th Elliptic Filter (uf=2)	16	52	68

(b)

Figure 9: (a) 5-stage adder with internal forwarding unit and 6-stage multiplier with internal forwarding unit (b) Characteristics of the benchmarks

Ben.	no-frw.		1 buf/in-frw.		in-frw.		1 buf/ex-frw.		2 buf/ex-frw.		ex-frw.	
	int/ aft	%	int/ aft	%	int/aft	%	int/aft	%	int/aft	%	int/aft	%
1	58/56	2	43/42	2	43/42	2	29/28	3	29/28	3	29/28	3
2	20/17	15	18/15	17	18/15	17	14/13	7	14/12	14	12/ 11	8
3	50/29	42	43/24	44	42/24	42	24/14	42	15/12	20	15/12	20
4	25/ 8	68	24/8	67	22/8	64	12/ 8	33	9/8	11	9/8	11
5	191/ 150	21	164/145	12	164/145	12	89/83	7	70/67	4	70/67	4
6	76/63	13	54/30	44	53/28	47	31/24	23	28/24	14	27/24	11
7	115/ 111	4	84/80	5	84/80	5	57/52	9	54/52	4	54/52	4

Table 1: Performance comparison for 1 adder and 1 multiplier system

cant criterion. Since most of the tested applications require more than one addition and one multiplication, increasing the number of functional units can reduce the underlying completion time. Doubling the number of adders and multipliers makes the initial schedule length shorter than that of the single functional unit version. According to the result presented in Table 1, for the system with an external forwarding hardware, processing a large application, such as the unfolded elliptic filter, the adder unit is occupied at almost every control step. Adding more functional units is the only approach that would reduce the schedule length. Table 2 shows the experimental results for the system with 2 multipliers and 2 adders.

According to the data from Table 2, even though we have added more functional units to the target system the hazard reduction percentage (ranging from 2–80 %) still relies on the characteristic of the applications as well as the pipeline architectures. For example, without the forwarding feature, in the lattice filter application, hazards can be reduced up to 45 percent in the 2-adder and 2-multiplier system. For the wave digital filter (benchmark 4), the reduction is 80%.

The experimental results from both tables show that SHARP can reduce a large number of hazards by considering all available hardware and overlapping pipeline instructions. Further, in each iteration of SHARP, the algorithm only needs to reschedule a few number of nodes. Our algorithm can also help designers choose the appropriate hardware architecture, such as the number of pipelines, pipeline depth, the number of forwarding buffers and others, in order to obtain good performance when running applications subject to their overhead hardware costs.

5 Conclusion

Since computation-intensive applications contain a significant number of data dependencies and few or no control instructions, data hazards often occur during the execution time which degrades the system performance. Hence, reducing the data hazards can dramatically improve the total computation time of such applications. Our algorithm, SHARP, supports modern multiple pipelined architectures and applies the loop pipelining technique to improve the system output. It takes the application characteristics in the form of a pipeline data-flow graph and target system information (e.g., the number of pipelines and depth, their associated types, and their forwarding buffer mechanism) as inputs. SHARP reduces data hazards by rearranging the execution sequence of the instructions and produces a schedule in accordance with the system constraints. Not only does SHARP serve as a scheduling optimization tool, it can be a simulation tool for a system designer as well.

References

[1] L. Chao, A. LaPaugh, and E. Sha. Rotation Scheduling: A Loop Pipelining Algorithm. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 566–572, June 1993.

[2] S. Davidson, D. Landskov, et al. Some Experiments in Local Microcode Compaction for Horizontal Machines. *IEEE Transactions on Computers*, c-30:460–477, July 1981.

[3] D. D. Gajski, N. D. Dutt, A. C-H Wu, and S. Y-L Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[4] J. L. Hennesy and D. A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan and Kaufmann Publisher Inc., California, 1990.

[5] P. D. Hoang and J. M. Rabaey. Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput. *IEEE Transactions on Signal Processing*, 41(6), June 1993.

[6] R. B. Jones and V. H. Allan. Software Pipelining: An Evaluation of Enhanced Pipelining. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, volume 24, pages 82–92, 1991.

[7] R. A. Kamin, G. B. Adams, and P. K. Dubey. Dynamic List-Scheduling with Finite Resources. In *International Conference on Computer Design*, pages 140–144, Oct. 1994.

[8] A. A. Khan, C. L. McCreary, and M. S. Jones. A Comparison of Multiprocessor Scheduling Heuristics. In *1994 International Conference on Parallel Processing*, volume II, pages 243–250. IEEE, 1994.

[9] P. M. Kogge. *The Architecture of Pipelined Computers*. McGraw-Hill, New York, 1981.

[10] M. Lam. Software Pipelining. In *Proceedings of the ACM SIGPLAN'88 Conference on Progaming Language Design and Implementation*, pages 318–328, June 1988.

Ben.	no-frw.		1 buf/in-frw.		in-frw.		1 buf/ex-frw.		2 buf/ex-frw.		ex-frw.	
	int/ aft	%	int/ aft	%	int/aft	%	int/aft	%	int/aft	%	int/aft	%
1	57/56	2	39/38	3	37/36	3	18/17	5	18/17	5	18/17	5
2	19/18	5	16/15	6	16/15	6	12/11	8	9/8	11	9/8	11
3	49/27	45	40/23	43	40/23	43	24/11	54	12/9	25	12/9	25
4	23/5	78	23/4	83	21/4	81	11/6	46	6/4	33	6/4	33
5	180/158	12	155/136	12	155/136	12	76/69	9	47/44	6	47/44	6
6	73/67	8	49/42	14	49/41	16	23/16	30	19/13	32	18/13	27
7	114/112	2	77/74	3	75/73	3	39/37	5	32/31	3	31/30	3

Table 2: Performance comparison for 2-adder and 2-multiplier system

[11] C. Leiserson and J. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, pages 5–35, June 1991.

[12] K. K. Parhi and D. G. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. In *Proceedings of the International Conference on Parallel Processing*, volume I, pages 209–216, 1989.

[13] B. Ramakrishna Rau. Iterative Modulo Scheduling: An Algorithm For Software Pipelining Loops. In *Proc. 27th Annual International Symposium on Microarchtecture*, pages 63–74, November 1994.

[14] B. Shirazi et al. PARSA: A PARallel program Scheduling and Assessment environment. In *International Conference on Parallel Processing*, 1993.

[15] S. Shukla and B. Little. A Compile-time Technique for Controlling Real-time Execution of Task-level Data-flow Graphs. In *1992 International Conference on Parallel Processing*, volume II, 1992.

[16] R. M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development*, 11(1):25–33, January 1967.

Sissades Tongsima was born in Bangkok, Thailand, on January 1, 1970. He received the B.Eng degree in Industrial Instrumental Engineering in 1991 from the King Mongkut Institute of Technology (Ladkrabang) Thailand. In 1992, he was awarded the Royal Thai Government Scholarships to pursue his study in Computer Science majoring in parallel and distributed computing area. He obtained his M.S. and Ph.D. degrees from the department of Computer Science and Engineering, University of Notre Dame in 1995 and 1999 respectively. Dr. Tongsima research interests while studying at Notre Dame include data scheduling and partitioning on parallel and distributed systems, high-level synthesis, loop transformations, rapid prototyping, and fuzzy systems. Upon the completion of his study, he came back to Thailand. Since June 1999, Dr. Tongsima has joined the National Electronics and Computer Technology Center (NECTEC). He is currently a researcher of the High Performance Computing Division at NECTEC.

Chantana Chantrapornchai received her Ph.D. degree in Computer Science and Engineering, University of Notre Dame, USA. in 1999. She received her bachelor degree in Computer Science from Thammasat University, Bangkok, Thailand in 1992 and M.S. degree in Computer Science from Northeastern University, Boston, Massachusetts in 1994. Currently Dr. Chantrapornchai is a faculty member in Department of Mathematics, Faculty of Science, Silpakorn University, Thailand. Her research interests include high-level synthesis, rapid prototyping, and fuzzy systems.

Edwin H.-M. Sha (S'88-M'92) received the M.A. and Ph.D. degree from the Department of Computer Science, Princeton University, Princeton, NJ, in 1991 and 1992, respectively. Since August 1992, he has been with University of Notre Dame, Notre Dame, IN. He is now an associate professor and the Associate Chairman of the Department of Computer Science and Engineering.

His research areas include Multimedia, Memory systems, Parallel and pipelined architectures, Loop transformations and parallelizations, Software tools for parallel and distributed systems, High-Level synthesis in VLSI, Fault-tolerant computing, VLSI processor arrays, and Hardware and software co-design.

He has published more than 100 research papers in referred conferences and journals during the past seven years. He is actively participating in professional activities. In 1994, He was the Program Committee Chair for the Fourth IEEE Great Lakes Symposium on VLSI and he served as the Program Committee Co-Chair for the 2000 ISCA 13th International Conference on Parallel and Distributed Computing Systems. He also served in program committee for many international conferences such as IEEE International Symposium on the Frontiers of Massively Parallel Computation,IEEE/ACM International Symposium on System Synthesis, and International Conference on Parallel and Distributed Computing and Systems, etc. He has been served as an associate editor for several journals. He received Oak Ridge Association Ju-

nior Faculty Enhancement Award in 1994, and NSF CAREER Award in 1995. He is also invited to be a guest editor for the special issue on Low Power Design of IEEE Transactions on VLSI Systems and is now serving as an editor for IEEE Transactions on Signal Processing. He received CSE Undergraduate Teaching award in 1998.

Nelson L. Passos: received the B.S. degree in electrical engineering, with specialization in telecommunications, from the University of Sao Paulo, Brazil, in 1974. He received the M.S. degree in Computer Science from the University of North Dakota, Grand Forks, ND, in 1992, and the Ph.D. degree in Computer Science and Engineering from the University of Notre Dame, IN in 1996. From 1974 to 1990, he worked at Control Data Corporation. Since 1996 he has been at Midwestern State University, Wichita Falls, TX, where he is currently an Associate Professor with the Department of Computer Science. His research interests include parallel processing, VLSI design, loop transformations, high-level synthesis, and multi-dimensional digital signal processing.

Dr. Passos was awarded the Professional Excellence and Bill Norris Shark Club Awards at Control Data Corporation. While with the Universoty of Notre Dame, he was awarded the William D. Mensch, Jr. Fellowship. He received an NSF grant award in 1997 on support to new research on multidimensional retiming.

แนะนำ UML เบื้องต้น

ดร.บรรจง หะรังษี และ นางญาณวรรณ สินธุภิญโญ
นักวิจัย งานโครงการเครือข่าย NECTECNet
ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

ABSTRACT -- UML is a new wave in the software industry that has continued to grow up day by day and year by year. Currently it is widely said among system analysts, programmers, software engineers and software-related people, of applying the UML methodology to office and organization applications. For those who are interested in the software engineering process, this tutorial article is aimed at introducing the basics of UML: What is UML? and What are the fundamental working mechanisms of UML? In addition it is highly hoped in the end that this article will do a certain degree, assist those people in making a decision whether to bring UML to use with your office or not.

บทคัดย่อ-- UML เป็นคลื่นลูกใหม่มาแรงในแวดวงอุตสาหกรรมซอฟต์แวร์ ซึ่งมีการเติบโตขึ้นเรื่อยๆ อย่างดีวันดีคืน ปัจจุบันในแวดวงนักวิเคราะห์ระบบ โปรแกรมเมอร์ หรือผู้ที่ทำงานเกี่ยวกับซอฟต์แวร์เป็นที่กล่าวกันอย่างแพร่หลายถึงการนำเอา UML มาประยุกต์ใช้กับระบบงานต่างๆ ในสำนักงานหรือองค์กรของตน บทความ Tutorial ฉบับนี้มีจุดมุ่งหมายที่จะแนะนำโดยเบื้องต้นว่า UML คืออะไรและมีกลไกการทำงานโดยพื้นฐานอย่างไร เพื่อให้บุคคลที่เกี่ยวข้องโดยเฉพาะทางด้านซอฟต์แวร์ได้ทราบและเรียนรู้ถึงภาพโดยรวมของ UML นอกจากนั้นแล้วผู้เขียนยังหวังว่าบทความนี้อาจจะมีส่วนช่วยในการตัดสินใจถึงการที่จะนำเอา UML มาใช้กับหน่วยงานของท่านหรือไม่

คำสำคัญ -- Software Methodology, Modeling Language, Object-Oriented Analysis and Design, UML, Object-Oriented Programming, Software Engineering, Software Development, System Analysis and Design

---

ผู้อ่านหลายท่านคงได้คุ้นหูถึงคำว่า UML หรือ Unified Modeling Language และคงไม่แน่ใจว่า UML คืออะไร จะนำไปใช้งานได้อย่างไร บทความนี้ผู้เขียนจึงอยากจะแนะนำให้ผู้อ่านรู้จักและเข้าใจถึง UML ในแง่มุมต่างๆ ดังนี้

- นิยามของ UML และทำไมเราต้องใช้ UML
- กลไกการทำงานเบื้องต้นของ UML ว่ามีองค์ประกอบอะไรบ้าง
- ภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML

1. UML คืออะไร

การดำเนินงานโครงการพัฒนาซอฟต์แวร์ประกอบด้วย การเก็บรวมรวมข้อมูลเกี่ยวกับความต้องการของผู้ใช้ (Requirement Collection) ในการใช้ระบบ การวิเคราะห์ข้อมูลเหล่านั้น (Analysis) การออกแบบ (Design) และการเขียน

โปรแกรมหรือการสร้างซอฟต์แวร์ (Implementation) เป็นงานศิลปะประเภทหนึ่ง ที่มีความละเอียดอ่อน เพราะต้องการความพิถีพิถันในทุกขั้นตอนนับตั้งแต่การวิเคราะห์ข้อมูลที่ได้รับจนกระทั่งการพัฒนาขึ้นมาเป็นซอฟต์แวร์ (ระบบ คือ ซอฟต์แวร์ที่เราจะทำการพัฒนาขึ้นมาใช้)

จุดนี้จึงเกิดเป็นคำถามว่าเราจะมีวิธีการหรือเครื่องมือที่ดีอันหนึ่งหรือไม่ ที่จะช่วยให้การดำเนินงานโครงการทำซอฟต์แวร์โครงการหนึ่งเป็นไปอย่างมีประสิทธิภาพ ประสิทธิผล และมีการประสานกลมกลืนนับตั้งแต่ต้นจนจบ คำตอบก็คือ "มี" และ UML ก็คือเครื่องมือซึ่งสามารถ *"ช่วย"* เราได้นับตั้งแต่การวิเคราะห์ การออกแบบ และการดำเนินการพัฒนา

ตัว UML จริงๆ แล้วไม่สามารถดำเนินการสร้างซอฟต์แวร์ได้ กล่าวคือ ไม่สามารถทำการสร้างโปรแกรมได้ (Code Generation) แต่ทว่าผลพวงภายหลังจากการออกแบบมีรูปแบบหรือหน้าตาที่โปรแกรมเมอร์สามารถที่จะดำเนินการพัฒนาโปรแกรม (Coding) ได้อย่างเร็วและง่ายดายมาก

UML มององค์ประกอบต่างๆ ของซอฟต์แวร์ที่จะทำการพัฒนาขึ้นมาในรูปของออบเจ็คต์ (Object) [1] และออบเจ็คต์แต่ละตัวนั้นมีความเกี่ยวข้องกันโดยอาศัยความสัมพันธ์(Relationships) เป็นตัวเชื่อมโยง อีกทั้งออบเจ็คต์เหล่านั้นสามารถติดต่อสื่อสารกันได้ การติดต่อสื่อสารกันระหว่างออบเจ็คต์นี้เองเป็นกลไกภายในซอฟต์แวร์ที่ทำให้ซอฟต์แวร์ทำงานตามที่ผู้ใช้ต้องการได้

จากการมองซอฟต์แวร์เป็นออบเจ็คต์ นี่เอง UML จึงช่วยให้การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) เป็นไปได้ง่าย เพราะการเขียนโปรแกรมเชิงวัตถุก็มององค์ประกอบของซอฟต์แวร์เป็นออบเจ็คต์เช่นเดียวกัน

1.1 ประวัติโดยย่อของ UML

อาจจะมีคำถามหนึ่งที่ว่า UML เป็นวิธีการเดียวหรือไม่ที่สามารถนำมาประยุกต์ใช้ในการดำเนินงานโครงการทำซอฟต์แวร์ คำตอบก็คือ ยังมีวิธีการอื่นๆ อีกหลายวิธีที่เกิดขึ้นมาในช่วงประมาณ 20 ปีที่ผ่านมา ซึ่งแต่ละวิธีมีทั้งจุดอ่อนและจุดแข็งที่แตกต่างกันไป แต่ทว่ามีอยู่ 4 วิธีการต่อไปนี้ซึ่งเป็นตัวที่เด่นและเป็นที่รู้จักกันดีในแวดวงอุตสาหกรรมซอฟต์แวร์

- วิธีการของ Booch
- วิธีการ Object Modeling Technique (OMT)
- วิธีการ Object-Oriented Software Engineering (OOSE)
- วิธีการของ Coad และ Yourdon

ในปี 1994 ผู้นำของ 3 ค่ายคือ Grady Booch (วิธีการของ Booch), James Rumbaught (วิธีการ OMT) และ Ivar Jacobson (วิธีการ OOSE) ได้ร่วมมือกันและเริ่มต้นกระบวนการผสานวิธีการทั้งสามเข้าเป็นหนึ่งเดียว (Unified) และจุดนี้จึงเกิดเป็นจุดกำเนิดของ ***Unified Modeling Language*** หรือ ***UML*** นั่นเอง

นับแต่นั้นเป็นต้นมาจึงมีการร่วมมือกันของกลุ่มบริษัทต่างๆ ทั้งในอุตสาหกรรมซอฟต์แวร์และคอมพิวเตอร์ โดยมีแกนนำคือ Rational Software Cooperation และในปี ค.ศ. 1997 UML เวอร์ชัน 1.1 ก็ถือกำเนิดขึ้นมาและได้รับการยอมรับให้เป็นมาตรฐานอุตสาหกรรมซอฟต์แวร์นับจากนั้นเป็นต้นมา

กลุ่มของบริษัทยักษ์ใหญ่ที่เข้ามารวมตัวกันเป็นสมาชิกในการร่วมพัฒนา UML ประกอบไปด้วย Microsoft, IBM, Oracle, Sun และ Compaq ซึ่งจะเห็นได้ว่าบริษัทเหล่านี้ล้วนมีชื่อเสียงในแวดวงอุตสาหกรรมซอฟต์แวร์และคอมพิวเตอร์ทั้งสิ้น

1.2 ความหมายของคำว่า “โมเดล”

ก่อนอื่นมารู้จักกับคำว่า “ปัญหา” ก่อน ในโครงการทำซอฟต์แวร์หนึ่งๆ นั้น แต่ละขั้นตอนของการดำเนินงานไม่ว่าจะเป็นการวิเคราะห์ ออกแบบ หรือพัฒนา ก็คือปัญหาที่เราต้องดำเนินการแก้ไข ดังนั้นเมื่อกล่าวถึงคำว่า “ปัญหา” ความ

หมายก็คือ สิ่งที่เราต้องดำเนินการแก้ไขจนกระทั่งโครงการทำซอฟต์แวร์เสร็จสิ้นการดำเนินงาน

คำว่า "โมเดล" ในวลี Modeling Language มีความหมายดังนี้

โมเดล คือ ความพยายามในการที่จะอธิบายปัญหาของซอฟต์แวร์ที่จะดำเนินการพัฒนาขึ้นมา ตัวโมเดลจะแสดงให้เห็นถึงออบเจ็คต์ต่างๆ ที่เกี่ยวข้อง และความสัมพันธ์ระหว่างออบเจ็คต์เหล่านั้น นอกจากนั้นโมเดลยังแสดงให้เห็นถึงวิธีการที่จะแก้ไขปัญหา เราอาจจะใช้ไดอะแกรม เนื้อความ (Text) หรือรูปแบบอื่นๆ ซึ่งเป็นที่ยอมรับกันระหว่างผู้พัฒนาและผู้ใช้ระบบในการนำเสนอโมเดลๆ หนึ่ง UML จะใช้ทั้งไดอะแกรมและเนื้อความเพื่อทำการนำเสนอโมเดลของมัน

ดังนั้นเมื่อกล่าวถึง Modeling Language ความหมายของมันก็คือ ภาษาที่เราเอาไว้อธิบายโมเดลนั่นเอง Modeling Language ทั้งหลายมักจะใช้ไดอะแกรมหรือเนื้อความในการอธิบายถึง ออบเจ็คต์และความสัมพันธ์ระหว่างออบเจ็คต์เหล่านั้น

UML เป็น Modeling Language ภาษาหนึ่งซึ่งสามารถใช้ในการแก้ไขปัญหาในการดำเนินงานโครงการซอฟต์แวร์ ในการแก้ไขปัญหาหนึ่งๆ UML จะใช้โมเดลที่มีรูปแบบแตกต่างกันจำนวนหนึ่งโดยแต่ละโมเดลจะมีมุมมอง (View) ของปัญหาในแง่ที่แตกต่างกันออกไป แต่เมื่อเอาโมเดลเหล่านั้นมาประกอบกันเข้า เราก็จะสามารถดำเนินการวิเคราะห์ ออกแบบ และพัฒนาซอฟต์แวร์ได้อย่างมีประสิทธิภาพ โมเดลที่ UML ใช้จะมีลักษณะต่อเนื่องกันไป กล่าวคือ โมเดลหนึ่งจะอาศัยโมเดลที่สร้างขึ้นมาก่อนหน้านี้เพื่อทำการสร้างโมเดลตัวต่อไป

2. ทำไมต้องใช้ UML

เนื่องด้วยซอฟต์แวร์ในปัจจุบันมีทั้งขนาดและความซับซ้อนที่มากยิ่งขึ้นและยิ่งขึ้นตามลำดับ การดำเนินงานโครงการซอฟต์แวร์ซึ่งเป็นงานที่ละเอียดอ่อนและสลับซับซ้อนจึงจำต้องมีเครื่องมือที่ดีอันหนึ่งเพื่อที่จะสามารถจัดการกับขนาดและความซับซ้อนของซอฟต์แวร์เหล่านั้นได้

วิธีการที่โปรแกรมเมอร์มักจะใช้กันอยู่เสมอคือ การกระโดดลงไปทำการพัฒนาโปรแกรม (Coding) เลย กล่าวคือ "คิดไปด้วยและทำการพัฒนาโปรแกรมไปด้วย" โดยปราศจากกระบวนการที่เป็นระบบระเบียบและผสานกลมกลืน นับตั้งแต่การวิเคราะห์ ออกแบบแล้วจึงมาทำการพัฒนาโปรแกรมนั่นเอง สิ่งนี้สามารถเปรียบเทียบได้กับการขึ้นสังเวียนชกมวยเลยโดยปราศจากการเรียนรู้เรื่องมวยอย่างถูกต้องและเหมาะสมก่อน

นอกจากเวลาที่ต้องเสียไปในการพัฒนาซอฟต์แวร์ที่มากกว่าปกติแล้ว ซอฟต์แวร์ที่ได้รับยังอาจเต็มไปด้วยบัก (Software Bugs) ซึ่งยากต่อการแก้ไข อีกทั้งเมื่อคำนึงถึงการนำซอฟต์แวร์เหล่านั้นมาใช้งานซ้ำ (Code Reuse) ในวันข้างหน้า หรือการจัดการกับซอฟต์แวร์เหล่านั้นเมื่อจำเป็นต้องมีเวอร์ชันถัดๆ ไป (Software Configuration Management) การดำเนินการจะกระทำได้ยากมาก

UML เป็นเครื่องมือที่ดีอันหนึ่งซึ่งจะทำให้ผู้ประกอบการซอฟต์แวร์สามารถดำเนินการทุกขั้นทุกตอนอย่างสมเหตุสมผลอย่างเป็นระบบระเบียบและมีความต่อเนื่องกันไปตั้งแต่ต้นจนจบการดำเนินงาน

3. กลไกการทำงานเบื้องต้นของ UML

จากที่ได้กล่าวเอาไว้ในตอนต้นว่า UML ประกอบด้วยโมเดลจำนวนหนึ่งที่นำเอามาใช้ร่วมกันเพื่อการดำเนินงานโครงการซอฟต์แวร์ โมเดลดังกล่าวคือ [2]

- Use Case Diagram
- Sequence Diagram
- Class Diagram
- Activity Diagram
- Collaboration Diagram
- Component Diagram
- Deployment Diagram

- Object Diagram
- Statechart Diagram

จากโมเดลต่างๆ ของ UML ที่แสดงให้ดูข้างบน ผู้เขียนเห็นว่าโมเดล 3 โมเดลแรก คือ Use Case Diagram , Sequence Diagram และ Class Diagram เป็นโมเดลพื้นฐานที่ใช้ในระบบงานทั่วไป พอเพียงสำหรับผู้อ่านที่จะทำความเข้าใจเบื้องต้นเกี่ยวกับกลไกการทำงานของ UML ส่วนโมเดลอื่นเป็นส่วนเพิ่มเติมที่ผู้ใช้สามารถเลือกมาใช้เพื่ออธิบายโมเดลที่ใช้ถ่ายถอดแนวความคิดของผู้พัฒนาหรือผู้วิเคราะห์ระบบ

3.1 Use Case Diagram

Use Case แปลตรงตัวมีความหมายว่า กรณีของการใช้งานที่เกิดจากมุมมองของผู้ใช้ระบบ ตัวอย่างง่ายๆ ของ Use Case ก็คือ

Create Order (ทำการลงใบสั่งซื้อสินค้า)
Modify Order (ทำการเปลี่ยนแปลงหรือแก้ไขใบสั่งซื้อสินค้า)
Delete Order (ทำการลบใบสั่งซื้อสินค้า)

หรืออาจจะมองได้ว่ากรณีการใช้งานดังกล่าว ก็คือการอธิบายฟังก์ชันการทำงานต่างๆของระบบนั่นเอง แต่ละกรณีข้างบนถือเป็นหนึ่งกรณีของการใช้งาน หรือเรียกทับศัพท์ว่าหนึ่ง Use Case ต่อจากนี้ไปผู้เขียนจะใช้คำว่า Use Case ทับศัพท์ไปเลย

โดยปกติแล้วเราสามารถสร้างแต่ละ Use Case ขึ้นมาโดยอาศัยการสัมภาษณ์จากกลุ่มผู้ใช้ระบบ กรณีที่เราไม่รู้ว่าใครคือกลุ่มคนเป้าหมายที่เราควรจะไปทำการสัมภาษณ์สามารถหาข้อมูลเหล่านี้ได้จากฝ่ายบริหาร ซึ่งจะชี้ให้เรารู้ว่าแผนกไหน กองไหน และใครที่เราควรจะไปติดต่อเพื่อทำการสัมภาษณ์ มองกันอย่างง่ายๆ กลุ่มคนเหล่านั้นที่เราจะไปทำการสัมภาษณ์แต่ละคน ก็คือ Actor หรือผู้ใช้ระบบนั่นเอง

Actor ในความหมายของ UML ก็คือ ผู้ที่ติดต่อกับระบบโดยอยู่ภายนอกระบบ Actor อาจจะขอให้ระบบทำอะไรให้สักอย่างหนึ่ง หรือในทางกลับกันระบบอาจจะขอให้ Actor นั้นทำอะไรให้สักอย่างหนึ่ง [3] ดังนั้นจริงๆ แล้ว Actor ยังหมายความรวมถึงสิ่งอื่นๆ ที่อยู่นอกระบบซึ่งสามารถทำการติดต่อกับระบบได้ อาทิเช่น ระบบสินค้าคงคลัง ระบบบัญชี ระบบพัสดุ เป็นต้น

มาดูกันที่ตัวอย่างของ Use Case เพื่อทำความเข้าใจหรือมองเห็นภาพได้ง่ายขึ้น ตัวอย่างในรูปที่ 1 คือ ตัวอย่างของ Use Case เพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเข้าไปในระบบ โดยแสดงเป็นรูปแบบ (Template) อย่างง่ายๆ รูปแบบหนึ่ง รูปแบบอย่างง่ายนี้สามารถใช้ในการนิยาม Use Case ทั่วๆ ไป รูปแบบดังกล่าวมีองค์ประกอบดังนี้

- ชื่อของ Use Case
- ภาพรวมของการทำงาน (Overview)
- Actor หลัก (Primary Actor)
- Actor รอง (Secondary Actor)
- จุดเริ่มต้น (Starting Point)
- จุดสิ้นสุด (End point)
- การทำงานของ Use Case (Flow of Events)
- การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events)
- ผลของการทำงานของ Use Case (Measurable Result)

แต่ละองค์ประกอบมีความหมายดังนี้

- ชื่อของ Use Case : เป็นการกำหนดชื่อของ Use Case ตัวนี้ ชื่อที่ใช้ควรสั้นและกระชับแต่สื่อความหมายที่ดีของ Use Case ตัวนี้
- ภาพรวมของการทำงาน (Overview) : อธิบายภาพการทำงานของ Use Case ตัวนี้โดยรวมว่าทำอะไร ส่วนนี้ควรมีความยาวไม่เกิน 5 บรรทัด
- Actor หลัก (Primary Actor) : ใครคือผู้ใช้ของ Use Case ตัวนี้ จากตัวอย่าง Use Case : Create Order ผู้ใช้คือ ตัวแทนฝ่ายขายสินค้า

รูปที่ 1 : ตัวอย่างของ Use Case เพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเข้าไปในระบบ

Use Case : Create Order

ภาพรวมของการทำงาน (Overview)

จุดประสงค์หลักของ Use Case นี้คือ เพื่อทำการลงข้อมูลในใบสั่งซื้อสินค้าจากลูกค้า

Actor หลัก (Primary Actor)

ตัวแทนฝ่ายขายสินค้า

Actor รอง (Secondary Actor)

ไม่มี

จุดเริ่มต้น (Starting Point)

Use Case ตัวนี้เริ่มต้นเมื่อ Actor ตัวแทนฝ่ายขายสินค้าขอให้ระบบทำการลงข้อมูลการสั่งซื้อสินค้า

จุดสิ้นสุด (End point)

คำขอเพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเสร็จสิ้นสมบูรณ์หรือไม่ก็ถูกยกเลิก

การทำงานของ Use Case (Flow of Events)

จาก User Interface บนจอเพื่อทำการลงข้อมูลการสั่งซื้อ Actor จะต้องทำการใส่ข้อมูลเกี่ยวกับการสั่งซื้อ เป็นต้นว่า ***วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (Required Date) ปริมาณที่ต้องการสั่งซื้อ (Quantity) ต้องการให้ส่งมอบสินค้าโดยบริษัทรับส่งสินค้าไหน (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress)*** หลังจากนั้นแล้ว Actor สามารถเลือกที่จะทำการบันทึกข้อมูลลงไปไว้ในฐานข้อมูล หรือยกเลิกการทำงานทั้งหมด ถ้า Actor เลือกทำการบันทึก ใบสั่งซื้อสินค้าใบใหม่ก็จะถูกสร้างขึ้นมาในฐานข้อมูล

การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events)

ถ้าไม่มีสินค้าที่ต้องการอยู่ในคลังสินค้า ระบบจะต้องแจ้งให้ Actor ทราบพร้อมกันนั้นก็ต้องยกเลิกการทำงานที่เหลือของ Use Case นี้

ผลของการทำงานของ Use Case (Measurable Result)

จะมีใบสั่งซื้อสินค้าใหม่ 1 ใบ ถูกสร้างขึ้นมาในระบบ

หมายเหตุ : ในแต่ละส่วนของ Use Case ข้างบน เราได้ใช้ภาษาอังกฤษประกอบการอธิบาย เพื่อให้รู้ที่มาว่าในภาษาอังกฤษเขาใช้ Term ต่างๆ กันอย่างไร แต่เมื่อมีการนำ Use Case ไปใช้งานจริง ไม่จำเป็นต้องมีภาษาอังกฤษอีกต่อไป

- Actor รอง (Secondary Actor) : ใครคือผู้ใช้อื่นๆ นอกเหนือจาก Actor หลัก ที่เป็นผู้ใช้ Use Case ตัวนี้ ตัวอย่าง Use Case : Create Order ข้างบนไม่มีผู้ใช้อื่นๆ ที่จะใช้ Use Case ตัวนี้ร่วมกับ Actor หลัก ในบางระบบงานอาจจะเป็นไปได้ว่ามีผู้ใช้อื่น เช่น ผู้จัดการฝ่ายขายซึ่งสามารถใช้ Use Case ตัวนี้ร่วมกับตัวแทนฝ่ายขายสินค้า
- จุดเริ่มต้น (Starting Point) : เป็นการอธิบายเงื่อนไขหรือข้อกำหนดต่างๆ ณ จุดเริ่มต้นของ Use Case โดยเงื่อนไขเหล่านี้ โดยปกติแล้วจะต้อง “มี” หรือ “เป็น” อย่างนั้นก่อนการทำงานของ Use Case นี้ ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เงื่อนไขคือ Use Case จะเริ่มต้นเมื่อ Actor ขอให้ระบบทำการลงข้อมูลการสั่งซื้อสินค้าในใบสั่งซื้อสินค้า
- จุดสิ้นสุด (End point) : คล้ายกับจุดเริ่มต้น เป็นการอธิบายเงื่อนไขหรือข้อกำหนดต่างๆ ณ จุดสิ้นสุดการทำงานของ Use Case ตัวนี้ ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เงื่อนไขที่จุดสิ้นสุดคือ คำขอเพื่อการลงบันทึกข้อมูลการสั่งซื้อสินค้าจะต้องเสร็จสิ้นสมบูรณ์หรือไม่ก็ถูกยกเลิก กล่าวคือสิ่งใดสิ่งหนึ่งเท่านั้นจะเกิดขึ้น แต่ไม่ใช่ทั้งสองเกิดขึ้นพร้อมๆ กัน
- การทำงานของ Use Case (Flow of Events) : อธิบายขั้นตอนของการทำงานของ Use Case นี้ ตัวอย่างของ Use Case ข้างบนแสดงให้เห็นขั้นตอนการทำงานของ Use Case นี้เพื่อทำการลงบันทึกข้อมูลการสั่งซื้อสินค้านับตั้งแต่เริ่มต้นจนกระทั่งสิ้นสุด ขั้นตอนการทำงานเหล่านี้เรียกว่า ขั้นตอนการทำงานปกติของ Use Case
- การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events) : การทำงานของ Use Case ในกรณีเมื่อมีปัญหาใดๆ ก็ตามเกิดขึ้น และทำให้การทำงานตามปกติของ Use Case ไม่สามารถดำเนินต่อไปได้ จะทำให้มีการส่งต่อการทำงานของ Use Case จากขั้นตอนการทำงานปกติ (ในส่วนของการทำงานของ Use Case) มาที่ส่วนนี้ของ Use Case ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เมื่อระบบพบว่าสินค้าตัวที่ลูกค้าต้องการไม่มีอยู่ในคลังสินค้า นั่นหมายถึงระบบมี*ปัญหา*เกิดขึ้น ระบบก็จะทำการแจ้งให้ Actor ทราบและยกเลิกการทำงานที่เหลือของ Use Case นี้ทันที
- ผลของการทำงานของ Use Case (Measurable Result) : ผลการทำงานของ Use Case แสดงให้เห็นว่าเมื่อเสร็จสิ้นการทำงานของ Use Case นี้แล้ว จะมีอะไรเกิดขึ้นเป็นผลพวง ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู จะมีใบสั่งซื้อสินค้าใบใหม่เพิ่มขึ้น 1 ใบในฐานข้อมูลของเรา

รูปแบบของ Use Case ที่แสดงให้ดูในรูปที่ 1 เป็นเพียงรูปแบบง่ายๆ รูปแบบหนึ่งเท่านั้น ส่วนการใช้งานจริงผู้ใช้อาจเพิ่มเติมส่วนต่างๆ ลงไปเองได้ตามความเหมาะสม ทั้งนี้ขึ้นอยู่กับการตกลงกันระหว่างกลุ่มผู้พัฒนาระบบและผู้ใช้ระบบนั้นๆ

3.2 Sequence Diagram

Sequence Diagram คือ ไดอะแกรมที่แสดงลำดับขั้นตอน (Sequence) การทำงานภายในของ Use Case ตัวหนึ่ง โดยตัว Use Case เองแล้วเราจะไม่สามารถมองเห็นลำดับขั้นตอนการทำงานภายในของ Use Case ตัวนั้นได้ ตัว Sequence Diagram ต่างหากที่ทำให้เราสามารถมองเห็นลำดับขั้นตอนการทำงานภายในของ Use Case ตัวนั้นได้

นอกจากนี้ Sequence Diagram ยังแสดงให้เห็นถึงการติดต่อกันระหว่าง

- ออบเจ็คต์ต่างๆ ของ Use Case นั้น และ
- ออบเจ็คต์ และ Actor ของ Use Case นั้น

การติดต่อกันดังกล่าวจะทำให้มีข้อความ (Message) วิ่งไปมาในไดอะแกรมนั้น

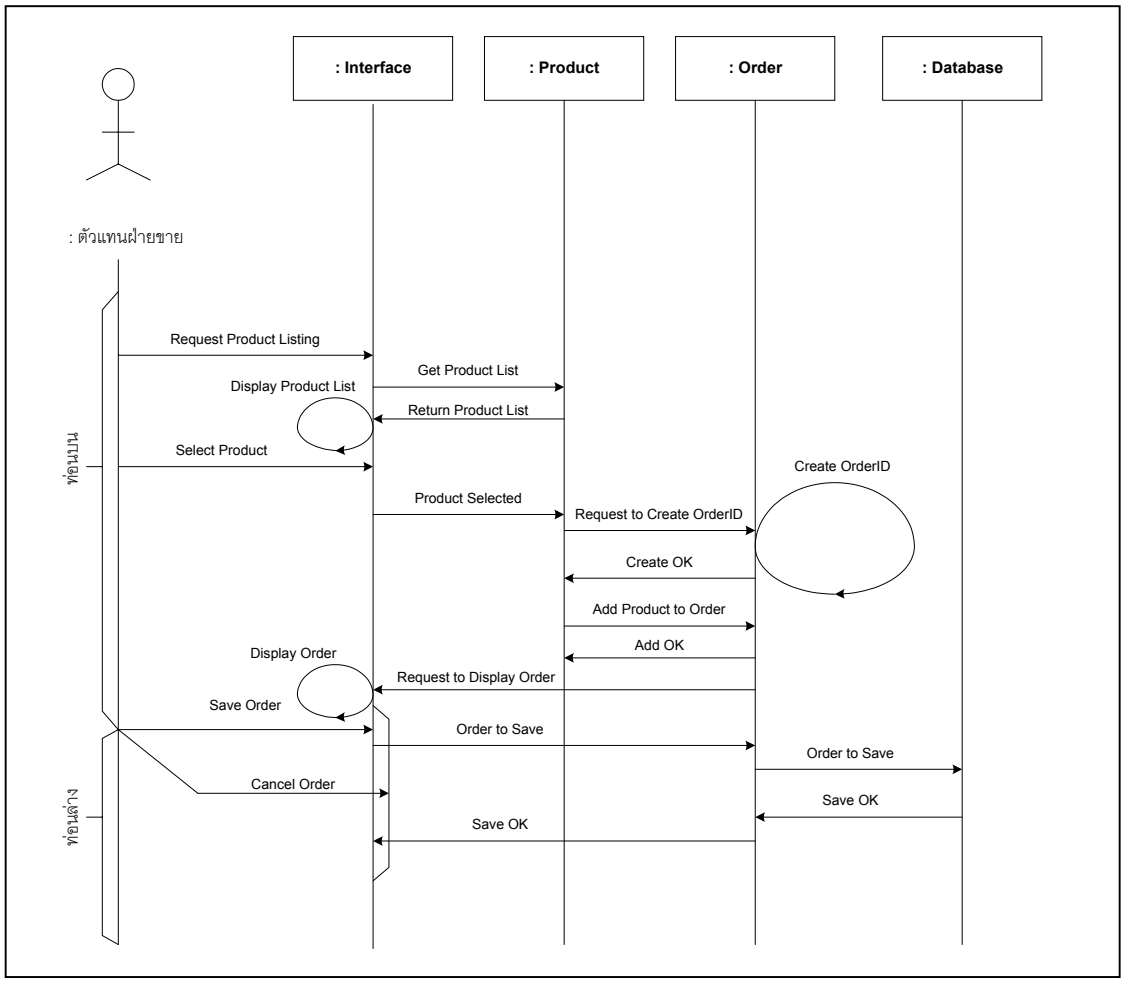
ที่กล่าวว่า “ออบเจ็คต์ต่างๆ ของ Use Case” สิ่งนี้หมายความว่าใน Use Case ตัวหนึ่งๆ จะมีออบเจ็คต์ที่เกี่ยวข้องกับ Use Case นั้นมากกว่า 1 ประเภท อาทิเช่น Use Case : Create Order อย่างน้อยจะประกอบไปด้วยออบเจ็คต์ 2 ประเภท กล่าวคือ Product และ Order

Product คือ สินค้าที่ลูกค้าต้องการซื้อ ส่วน Order คือ ใบสั่งซื้อสินค้าใบหนึ่งของลูกค้า ดังนั้นจึงสามารถกล่าวได้ว่าออบเจ็คต์ Product และ Order ก็คือ “ออบเจ็คต์ของ Use Case นี้”

รูปที่ 2 แสดงให้เห็นถึง Sequence Diagram ของ Create Order ซึ่งจะเห็นได้ว่ามี ออบเจ็คต์ที่เกี่ยวข้อง 4 ประเภท กล่าวคือ Interface, Product, Order และ Database (สังเกตุกล่องสี่เหลี่ยมข้างบนไดอะแกรม) ส่วนรูปตัวคนทางซ้ายมือก็คือ Actor ตัวแทนฝ่ายขาย (ซึ่งก็คือ Actor ของ Use Case: Create Order นั่นเอง)

แกนในแนวตั้ง 5 แกนคือ แกนเวลาของ ไดอะแกรม ลูกศรในไดอะแกรมคือการส่งข้อความจากออบเจ็คต์หนึ่งไปหาออบเจ็คต์อีกตัวหนึ่ง เป็นต้นว่าข้อความ Get Product List ซึ่งส่งจากออบเจ็คต์ Interface ไปยังออบเจ็คต์ Product ข้อความที่อยู่ด้านบนของไดอะแกรมจะเป็นข้อความที่เกิดก่อนข้อความที่อยู่ด้านล่างเรียงตามลำดับเวลา เป็นต้นว่า ข้อความ Get Product List จะเกิดขึ้นก่อนข้อความ Return Product List ดังนั้นจึงสามารถกล่าวได้ว่า Sequence Diagram เป็นไดอะแกรมที่เน้นเรื่องการเกิดขึ้นเรียงตามลำดับเวลา (Temporal)

รูปที่ 2 Sequence Diagram ของ Create Order



ของข้อความต่างๆ ภายใน ไดอะแกรม

ก่อนอธิบายตัว Sequence Diagram สำหรับ Create Order ขออธิบายข้อจำกัดของ Create Order ที่เราใช้เป็นตัวเดินเรื่องทั้งหมดก่อน เพื่อจุดประสงค์ในการแสดงให้ดูเป็นตัวอย่าง การลงบันทึกในใบสั่งซื้อนี้จึงเป็นการลงบันทึกแบบง่ายๆ กล่าวคือ สามารถมีสินค้าในใบสั่งซื้อหนึ่งใบได้เพียงประเภทเดียว (ในทางปฏิบัติอาจจะมีหลายประเภทสินค้าในหนึ่งใบสั่งซื้อได้) ดังนั้นตัวอย่าง Sequence Diagram สำหรับ Create Order จะแสดงให้เห็นถึงลำดับของเหตุการณ์การลงบันทึกแบบง่ายๆ ดังกล่าว

Sequence Diagram สำหรับ Create Order มีการทำงานเรียงตามลำดับเวลาดังนี้

1. Request Product Listing : จาก User Interface ที่แสดงบนหน้าจอ Actor จะทำการขอให้ระบบแสดงรายการของสินค้าทั้งหมดออกมาทางหน้าจอ
2. Get Product List : ออบเจ็คต์ User Interface หรือเรียกง่ายๆ ว่า Interface ก็จะทำการส่งต่อคำขอนั้นไปให้ “เจ้าของเรื่อง” ซึ่งก็คือ Product
3. Return Product List : “เจ้าของเรื่อง” ออบเจ็คต์ Product ก็จะทำการส่งรายการสินค้าทั้งหมดที่มีอยู่กลับคืนมาให้ Interface
4. Display Product List : ออบเจ็คต์ Interface เมื่อได้รับรายการสินค้าแล้วก็จะทำการแสดงรายการของสินค้าเหล่านั้นออกมาทางหน้าจอให้ Actor ดู ซึ่งจะเห็นได้ว่าเป็นการส่งข้อความเข้าหาตัวเอง (Reflexive Message)
5. Select Product : Actor จะทำการคลิกที่สินค้าตัวหนึ่ง (จากรายการสินค้าทั้งหมดที่แสดงบนหน้าจอ) ซึ่งลูกค้าต้องการซื้อ
6. Product Selected : Interface ก็จะทำการส่งต่อข้อมูลสินค้าที่ Actor ทำการเลือกแล้วนั้นไปยัง Product
7. Request to Create OrderID : เมื่อ Product ได้รับข้อมูลสินค้านั้นแล้ว ก็จะส่งคำขอไปยัง Order เพื่อขอให้สร้าง ID ใหม่ให้ตัวหนึ่งสำหรับใบสั่งซื้อสินค้าใบนี้
8. Create OrderID : เจ้าของเรื่องคือ Order ก็จะทำการสร้าง ID ใหม่ขึ้นมา ID หนึ่งสำหรับใบสั่งซื้อนี้ ซึ่งจะเห็นได้ว่าเป็นการส่งข้อความเข้าหาตัวเอง (Reflexive Message) เช่นเดียวกับ Display Product List
9. Create OK : หลังจากนั้นออบเจ็คต์ Order ก็ส่งข้อความไปบอก Product ว่าคำขอเพื่อสร้าง ID นั้นได้ทำสำเร็จแล้ว
10. Add Product to Order : หลังจากนั้น Product จึงทำการส่งข้อมูลของสินค้า (ที่ลูกค้าต้องการซื้อ) ไปให้ Order เพื่อทำการลงบันทึกในใบสั่งซื้อสินค้านั้น
11. Add OK : เมื่อเสร็จสิ้นการบันทึกแล้ว Order จึงบอกให้ Product รู้ว่าได้ดำเนินการเสร็จแล้ว
12. Request to Display Order : พร้อมกันนั้น Order ก็ส่งคำขอไปยัง Interface เพื่อขอให้ Interface ทำการแสดงข้อมูลรายละเอียดทั้งหมดของใบสั่งซื้อนี้ออกมาทางหน้าจอเพื่อให้ Actor ดู
13. Display Order : Interface ทำการแสดงข้อมูลรายละเอียดของใบสั่งซื้อนี้ออกทางหน้าจอ (ตามคำขอจาก Order)
14. Save Order : ต่อจากนั้น โดย Interface Actor ก็จะทำการคลิกปุ่ม Save เพื่อทำการ Save ข้อมูลของใบสั่งซื้อสินค้านั้น
15. Cancel Order : ให้สังเกตในไดอะแกรมที่ข้อความ Save Order ที่ส่งจากตัวแทนฝ่ายขายจะมีข้อความอีกอันหนึ่งคือ ข้อความ Cancel Order แยกออกมาจากจุดร่วมเดียวกัน จุดที่มีการแยกออกดังกล่าวเป็นสัญญลักษณ์ของการตัดสินใจที่จะเลือกทำอันใดอันหนึ่งเท่านั้น ความหมายของจุดแยกนี้ก็คือ ถ้า Actor ตัดสินใจทำการ Save ขั้นตอนการทำงานที่เหลือเรียงตามลำดับเวลาจะประกอบไปด้วยข้อความ

Order to Save (จาก Interface): Interface ส่งข้อมูลของใบสั่งซื้อนี้ไปให้เจ้าของเรื่อง Order

Order to Save (จาก Order): เจ้าของเรื่อง Order ส่งต่อข้อมูลนั้นให้ Database เพื่อจะได้ทำการบันทึกข้อมูลนั้นต่อไป

Save OK (จาก Database): Database แจ้งให้ Order รู้ว่าได้ทำการบันทึกเรียบร้อยแล้ว

Save OK (จาก Order): Order แจ้ง Interface ว่าการบันทึกข้อมูลได้ดำเนินการเสร็จสิ้นแล้ว

ถ้าหากตัดสินใจยกเลิก (อาจเป็นเพราะราคาสินค้าที่ลูกค้าคิดว่าแพงเกินไป) คือ ไม่ Save การทำงานจะสิ้นสุดทันที โปรดสังเกตุว่าการตัดสินใจเลือกทำการ Save หรือ Cancel เป็น Statement หนึ่งที่ปรากฏในส่วนของการทำงานของ Use Case ด้วย (ให้ดูในส่วนของการทำงานของ Use Case ของ Use Case: Create Order ด้วย)

3.2.1 ความสอดคล้องกันระหว่าง Use Case และ Sequence Diagram

Sequence Diagram ในรูปที่ 2 จริงๆ แล้วถูกสร้างขึ้นมาจาก Use Case: Create Order (เนื่องจากบทความนี้เป็นบทความที่แสดงให้เห็นภาพโดยรวมของ UML จึงยังไม่สอนวิธีการสร้าง Sequence Diagram จาก Use Case) ในเบื้องต้นนี้เราจะชี้ให้เห็นถึงความสอดคล้องกันระหว่าง Use Case และ Sequence Diagram สำหรับ Create Order

ท่อนบนของ Sequence Diagram (หาคำว่า " ท่อนบน" ในไดอะแกรม) สำหรับ Create Order เทียบเท่ากับ Statement ดังนี้ (ให้ดูในส่วน**การทำงานของ Use Case** ของ Use Case: Create Order ในรูปที่ 1 ประกอบด้วย)

"จาก User Interface บนจอเพื่อทำการบันทึกข้อมูลการสั่งซื้อ Actor จะต้องทำการใส่ข้อมูลเกี่ยวกับการสั่งซื้อ เป็นต้นว่า วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (RequiredDate) ปริมาณที่ต้องการสั่งซื้อ (Quantity) ต้องการให้ส่งมอบสินค้าโดยบริษัทรับส่งสินค้าไหน (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress)"

ท่อนล่างของ Sequence Diagram (หาคำว่า " ท่อนล่าง" ในไดอะแกรม) เทียบเท่ากับ Statement ส่วนที่เหลือของ Use Case นี้ กล่าวคือ "หลังจากนั้น Actor สามารถเลือกที่จะทำการบันทึกข้อมูลลงไปไว้ในฐานข้อมูล หรือยกเลิกการทำงานทั้งหมด ถ้า Actor เลือกทำการบันทึกข้อมูล ใบสั่งซื้อสินค้าใบใหม่ก็จะถูกสร้างขึ้นมาในฐานข้อมูล"

ดังนั้น ณ จุดนี้จึงขอสรุปเกี่ยวกับ Sequence Diagram ไว้ดังนี้

Sequence Diagram ถูกสร้างขึ้นมาจาก Use Case ตัว Use Case มิได้บอกรายละเอียดของการทำงานภายในว่าเป็นอย่างไร Sequence Diagram ต่างหากเป็นตัวบอกว่ามีรายละเอียดการทำงานภายในอย่างไร และรายละเอียดการทำงานภายในเหล่านั้นก็คือ ข้อความที่วิ่งไปมาในไดอะแกรมนั้น

ส่วนคำถามถัดมาที่ผู้อ่านอาจอยากจะถามคือ ข้อความที่วิ่งไปมาเหล่านั้นมีประโยชน์อะไร เราจะไขคำตอบสำหรับคำถามนี้ในหัวข้อ Class Diagram ข้างล่าง

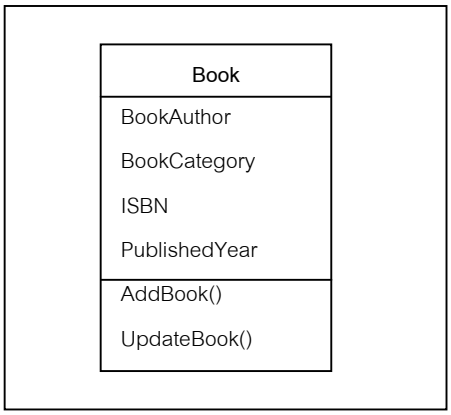
3.3 Class Diagram

ก่อนอื่นขอนิยามคำว่า "Class" ก่อน "Class" คือ เซ็ตของออบเจ็คต์ที่มีคุณสมบัติ (Properties) และพฤติกรรม (Methods) เช่นเดียวกัน ตัวอย่างเช่น Class ของหนังสือมีลักษณะดังแสดงในรูปที่ 3 และเราอาจมีออบเจ็คต์ต่างๆ ดังนี้อยู่ใน Class Book ดังตารางที่ 1 ข้างล่าง

ในตารางจะเห็นได้ว่าทั้ง 3 ออบเจ็คต์ที่แสดงในตารางก็คือ เซ็ทของออบเจ็คต์ที่มีคุณสมบัติ BookAuthor, BookCategory, ISBN และ PublishedYear เหมือนกัน แต่เนื้อหาสาระ (Content) ของคุณสมบัติเหล่านั้นจะไม่เหมือนกัน

เช่น หนังสือที่มี ISBN 1111 ผู้แต่ง คือ รัชนี ส่วนหนังสือที่มี ISBN 1234 ผู้แต่ง คือ ชาลี

รูปที่ 3 ตัวอย่าง Class Book



ตารางที่ 1 ตัวอย่างของ Class Book

Book-Author	Book-Category	ISBN	Published-Year
รัชนี	Algorithm	1111	1998
ชาลี	OS	1234	1995
นพพร	DBMS	1345	1996

นอกจากการมีคุณสมบัติที่เหมือนกันแล้ว ออบเจ็คต์ในคลาสเดียวกันยังมีพฤติกรรมที่เหมือนกันอีกด้วย ตัวอย่างเช่น ในการเปลี่ยนแปลงรายละเอียดของหนังสือเล่มหนึ่งๆ ออบเจ็คต์ทั้งหลายที่อยู่ใน Class นี้สามารถใช้พฤติกรรม UpdateBook() เพื่อทำการเปลี่ยนแปลงรายละเอียดของหนังสือเล่มนั้นๆ

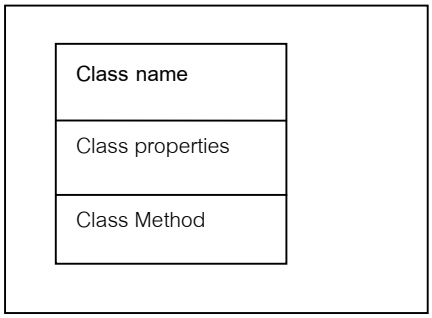
จากความหมายของ Class ที่ได้นิยามไว้ข้างบน ความหมายของ Class Diagram ก็คือภาพที่ประกอบไปด้วย Class ต่างๆ ที่มาประกอบหรือรวมตัวกันกลายเป็นระบบหรือซอฟต์แวร์ตัวหนึ่ง เป็นต้น ยกตัวอย่างเช่น Class : Interface, Product, Order, Database เป็นต้น

โดย Class ทั้งสี่ส่วนนี้มีที่มาจาก Sequence Diagram สำหรับ Create Order (ให้สังเกตุว่ากล่องสี่เหลี่ยมข้างบน Diagram คือ ออบเจ็คต์ของคลาสทั้งสี่นั่นเอง)

ในการดำเนินการพัฒนาซอฟต์แวร์ตัวหนึ่งๆ โปรแกรมเมอร์จะทำการพัฒนาซอฟต์แวร์ตาม Class ต่างๆ เหล่านั้นทั้งหมดที่ได้กำหนดขึ้นมา

Class Diagram ใน UML มีรูปแบบดังรูปที่ 4 คือ

รูปที่ 4 รูปแบบของ Class Diagram



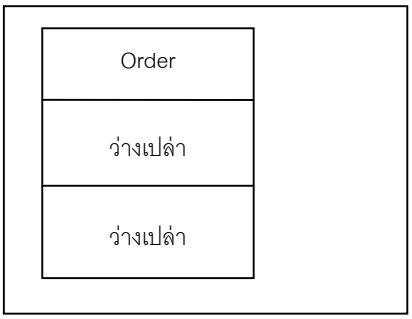
ซึ่งจะเห็นได้ว่าแต่ละส่วนของไดอะแกรมนี้ (ซึ่งมีอยู่ 3 ส่วน) จะสอดคล้องกับ Class Book ที่แสดงในรูปที่ 3

ในทางปฏิบัติระบบที่พัฒนาจะมี Class ต่างๆที่เกี่ยวข้องมากกว่า 1 Class เช่น ระบบการสั่งซื้อสินค้า อาจจะมี Class customer , Class Order ซึ่ง Class ทั้งสองอาจมีความสัมพันธ์กันในลักษณะใดลักษณะหนึ่ง แต่ในบทความฉบับนี้ต้องการเน้นให้เห็นถึงสัญญลักษณ์ของ Class Diagram และองค์ประกอบของ Class จึงยังไม่กล่าวถึงความสัมพันธ์ของ Class เหล่านั้น

เราจะใช้ตัวอย่างการทำการบันทึกข้อมูลเกี่ยวกับการสั่งซื้อสินค้า Create Order ในสองหัวข้อที่แล้วกล่าวคือ Use Case และ Sequence Diagram เพื่อแสดงให้ผู้อ่านเห็นถึงการได้มาซึ่ง Class Diagram สำหรับ Class Order (ส่วน Class อื่นๆ ก็จะอาศัยวิธีการเดียวกัน เพื่อจะให้ได้มาซึ่ง Class Diagram ของมัน)

เมื่อตอนเริ่มต้นสิ่งแรกที่เรามีสำหรับ Class Order คือ Class Order ซึ่งว่างเปล่าดังแสดงในรูปที่ 5

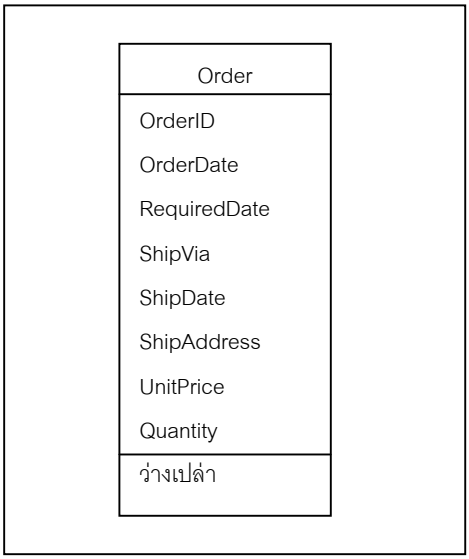
รูปที่ 5 ตัวอย่าง Class Order ที่ว่างเปล่า



3.3.1 Class Properties

ในส่วนของ Class Properties คุณสมบัติที่เกี่ยวข้องกับ Class Order มีดังนี้

รูปที่ 6 ตัวอย่าง Class Order



คุณสมบัติดังกล่าวได้มาจาก Use Case : Create Order โดยสังเกตุจากตัวอักษรเข้มใน **"ส่วนของการทำงานของ Use Case"** ความจริงอย่างหนึ่งที่เรารู้ก็คือ ในขณะที่เราทำการสัมภาษณ์ Actor ของ Use Case นี้ Actor จะเล่าให้เราฟังว่า

"เมื่อมีลูกค้าโทรเข้าสั่งของ ลูกค้าจะต้องการบอกเราในรายละเอียด เช่น วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (RequiredDate) ต้องการสินค้านั้นกี่ชิ้น (Quantity) ต้องการให้ส่งมอบสินค้าอย่างไร (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress) เป็นต้น"

Actor โดยปกติจะต้องมีรายละเอียด เช่น ราคาของสินค้าต่อชิ้น (UnitPrice) วันที่ที่สินค้าถูกส่งออกไปโดยบริษัทที่รับส่งของ (ShipDate) ส่วนคุณสมบัติของ OrderID ซอฟต์แวร์ที่จะดำเนินการสร้างจะทำการคำนวณให้เราโดยอัตโนมัติ ดังนั้นโดยสรุปแหล่งที่มาของคุณสมบัติของ Class หนึ่งๆ จะมาจาก Use Case (อาจจะมากกว่า 1 Use Case เช่น นอกจากมาจาก Create Order แล้วยังสามารถมาจาก Modify Order หรือ Delete Order ได้ด้วย) และข้อมูลจากการสัมภาษณ์กับ Actor ของ Use Case นั้น

3.3.2 Class Methods

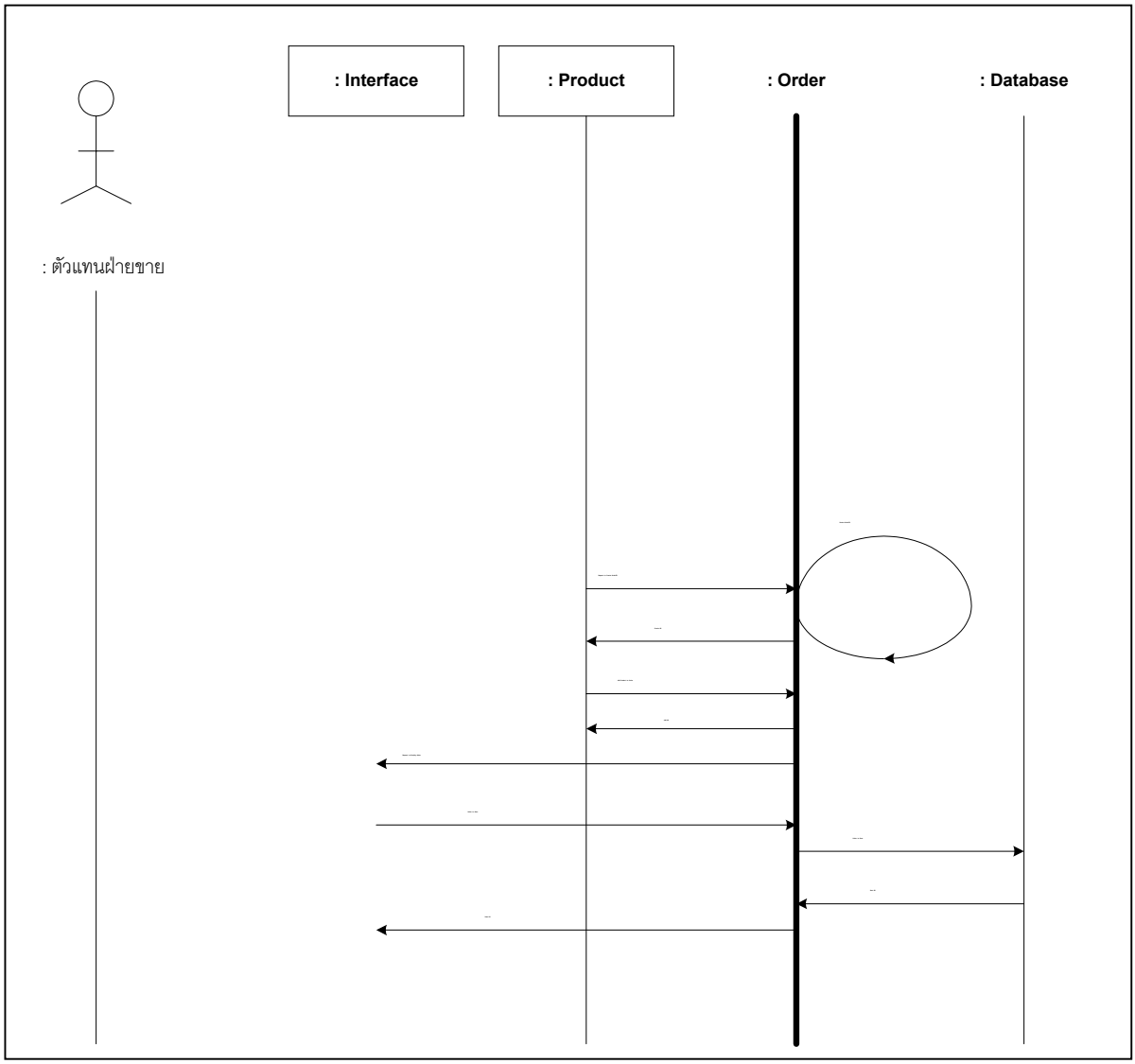
สิ่งสุดท้ายที่เราต้องการคือ Class Methods ซึ่งจะได้มาจาก Sequence Diagram สิ่งที่เราควรทราบเบื้องต้นก็คือ ข้อความแต่ละข้อความใน Sequence Diagram สามารถเทียบเท่ากับ Method 1 Method มาดูกันที่ตัวอย่างการกำหนด Class Methods ของ Class Order เพื่อจะได้เห็นภาพที่แท้จริงกันเลย

หลักการหา Method มีดังนี้

1. จาก Sequence Diagram สำหรับ Create Order พิจารณาเฉพาะข้อความที่วิ่งเข้าหรือออกจากแกนเวลาของ Order เท่านั้น รูปที่ 7 เป็นไดอะแกรมที่ตัดออกมาจากไดอะแกรมตัวสมบูรณ์ (ในรูปที่ 2) ไดอะแกรมที่ตัดออกมานี้ประกอบด้วยข้อความเฉพาะที่เกี่ยวข้องกับ Class Order เท่านั้น ซึ่งจะเห็นได้ว่ามีลูกศรวิ่งเข้าออกจากแกนเวลาของ Order

สิ่งหนึ่งที่ผู้อ่านอาจจะรู้สึกว่าน่าจะเป็นคือ ข้อความต่างๆที่วิ่งไปมาแต่ละข้อความน่าจะมี Method ตัวหนึ่งมาทำหน้าที่ในการสื่อสารนั้น ยกตัวเอย่างเช่น ข้อความ Request to Create OrderID จาก Product ไปหา Order จะมี Method : RequestToCreateOrderID() มาทำหน้าที่ในการสื่อสาร ซึ่งในที่นี้ก็คือ คำขอให้ Order ช่วยสร้าง ID ให้ 1 ID สำหรับใบสั่งซื้อสินค้านี้

รูปที่ 7 Sequence Diagram สำหรับ Create Order เฉพาะข้อความที่เกี่ยวข้องกับ Class Order



ณ จุดนี้คำถามหนึ่งที่ผู้อ่านอยากจะถามก็คือ ข้อความ Request to Create Order ID ควรจะเป็นของ Class ไหน ระหว่าง Class Product และ Order คำตอบของคำถามนี้จึงเกิดเป็นหลักการข้อที่ 2

2. หลักการในข้อนี้เป็นหลักการอันหนึ่งที่สามารถใช้ได้และเป็นที่ยอมรับโดยทั่วไป

"ถ้าข้อความที่พิจารณามีเนื้อหาสาระหนัก (weight) ไปทาง Class ไหนมากกว่า (ระหว่าง 2 Class) ข้อความนั้นก็จะกลายเป็น Method หนึ่ง Method ของ Class นั้น"

โดยใช้หลักการนี้ เราสามารถทำการกำหนด Method ให้กับ Class Order ได้ดังตารางที่ 2

เราสรุปได้ว่า Methods ของ Class Order ประกอบไปด้วย

- Create OrderID
- Create OK
- Add OK
- Request to Display Order
- Order to Sace (จาก interface)
- Save Order (จาก Order)

ตารางที่ 2 ตารางแสดงข้อความที่เกี่ยวข้องกับคลาสหนึ่งๆ

Message	Class			
	Interface	Product	Order	Database
Request to Create Order ID		✓		
Create Order ID			✓	
Create OK			✓	
Add Product to Order		✓		
Add OK			✓	
Request to Display Order			✓	
Order to Save (จาก Interface)	✓			
Order to Save (จาก Order)			✓	
Save OK (จาก Database)				✓
Save OK (จาก Order)			✓	

(ให้สังเกตุเครื่องหมาย ✓ ในคอลัมน์ Order ในตารางที่ 2)

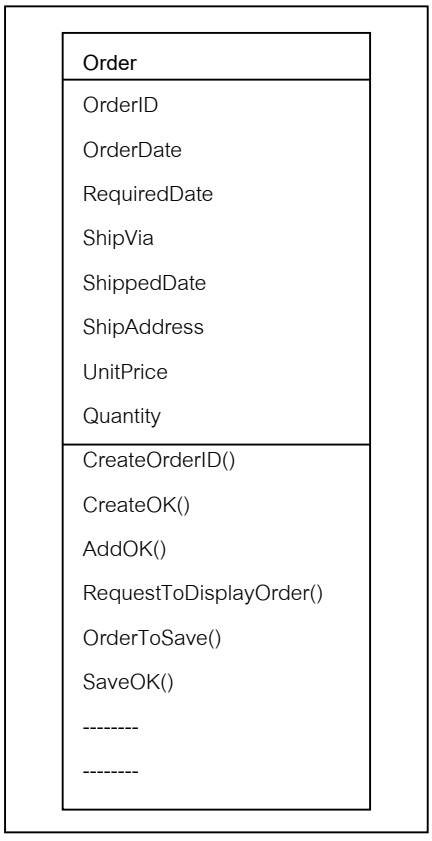
มาดูกันที่ Method อื่นๆ ซึ่งไม่อยู่ใน Class Order กันก่อน สาเหตุของการที่ Methodอื่นๆ (ให้ดูในตารางที่ 2 ประกอบตรงเครื่องหมาย ✓ ที่ไม่อยู่ในคอลัมน์ Order) กล่าวคือ

- RequestToCreateOrderID()
- AddProductToOrder()
- OrderToSave()
- SaveOK() (จาก Database)

Methods ของ Class Order สามารถอธิบายได้ดังนี้

- RequestToCreateOrderID() : ควรจะเป็นของ Product เพราะ Product เป็นคนขอให้ Order ทำการสร้าง OrderID ใหม่ขึ้นมาตัวหนึ่ง
- AddProductToOrder() : ควรจะเป็นของ Product เพราะProduct เป็นคนขอให้ Order ทำการใส่ข้อมูลของสินค้าที่ลูกค้าต้องการซื้อลงไปในใบสั่งซื้อสินค้านั้น
- OrderToSave() : ควรจะเป็นของ Interface เพราะ Interface เป็นคนส่งข้อมูลของใบสั่งซื้อสินค้าไปให้ Order

รูปที่ 8 Class Order และ Method



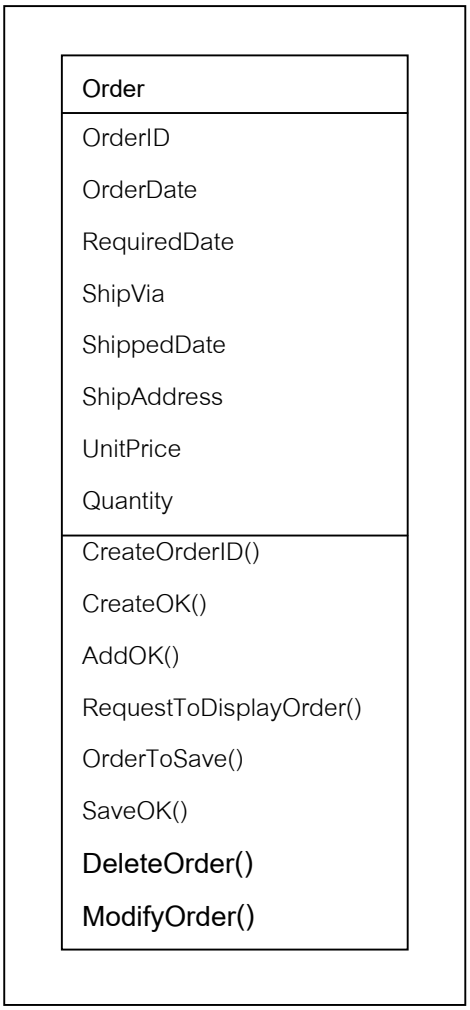
- SaveOK() (จาก Database) : ควรจะเป็นของ Database เพราะ Database เป็นคนแจ้งให้ Order ทราบว่าใบสั่งซื้อสินค้านั้นได้ทำการ Save เรียบร้อยแล้ว

สรุปโดยหลักการก็คือ ถ้า Class 1 เป็นผู้ขอให้ Class 2 ทำอะไรให้ก็ตาม (โดยการส่งข้อความไปให้ Class 2) Class 1 จะมี *น้ำหนักมากกว่า* ทั้งนี้เพราะ Class 1 เท่านั้นถึงจะที่รู้ดีว่ากำลังให้ Class 2 ทำอะไรให้อยู่ ดังนั้น Method ทั้งสี่ข้างบนจึงควรไปอยู่กับ Class อื่นๆ เหล่านั้นที่มิใช่ Order

ส่วน Method ที่เป็นของ Class Order ก็สามารถตอบได้โดยอาศัยหลักการเดียวกับหลักการในย่อหน้าที่แล้ว

3.3.3 Method ของ Class Order จาก Sequence Diagram อื่นๆ

รูปที่ 9 Class Order และ Method



Class Order อาจจะมี Sequence Diagram อื่นๆ (นอกจาก Create Order Sequence Diagram) ที่เกี่ยวข้องกับมัน อาทิเช่น Delete Order Sequence Diagram หรือ Modify Order Sequence Diagram เป็นต้น Sequence Diagram อื่นๆ เหล่านี้ก็จะเป็นที่มาของ Method อื่นๆ (นอกเหนือจาก Mehtod ในรูปที่ 8) เป็นต้นว่า DeleteOrder(), ModifyOrder() , และอื่นๆ ดังนั้น Class Diagram สำหรับ Order จึงสามารถรวมเอา Methods อื่นๆ เหล่านั้นเข้าไปด้วย กล่าวคือ (ให้ดูที่ Method ที่เป็นตัวอักษรเข้ม)

3.3.4 Class อื่นๆ นอกจาก Class Order

สำหรับ Class อื่นๆ เช่น Class : Interface, Product, Database เป็นต้น ผู้อ่านสามารถใช้วิธีการเดียวกันที่นำเสนอมาตั้งแต่ต้นในหัวข้อ Class Diagram เพื่อทำการหาคุณสมบัติและ Method ของมัน ในที่สุดแล้วเราก็จะได้ไดอะแกรมของทุก Class ในระบบของเราออกมาและสามารถนำเอาไปทำการพัฒนาเป็นโปรแกรมได้ต่อไป

4. ภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML

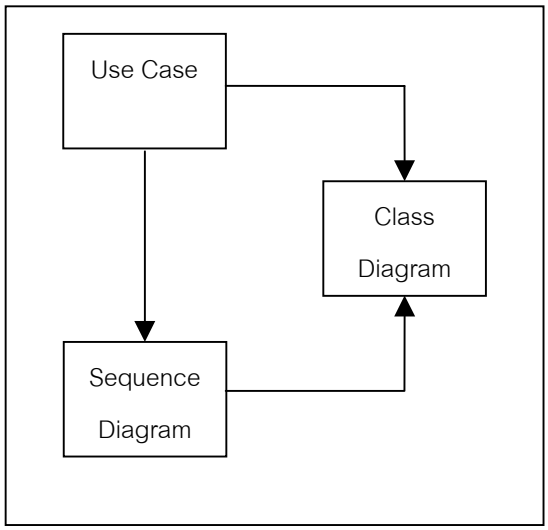
จากในหัวข้อ 3.1, 3.2 และ 3.3 ที่ได้นำเสนอไปแล้วในเรื่อง Use Case, Sequence Diagram และ Class Diagram เรียงตามลำดับหัวข้อดังกล่าว เราจึงสามารถสรุปภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML ดังแสดงไว้ในรูปที่ 10

กล่าวคือ

- สิ่งแรกที่เราต้องทำคือ การหา Use Case ทั้งหมดในระบบ
- ถัดมาคือ การสร้าง Sequence Diagram ที่สอดคล้องกับ Use Case ตัวหนึ่งๆ
- สิ่งสุดท้ายคือการสร้าง Class Diagram Class Diagram จะประกอบไปด้วย Class ทั้งหมดในระบบ ส่วน Class หนึ่ง Class ในไดอะแกรมนั้นโดยปกติจะถูกสร้างขึ้นมาจากหลาย Use Case และหลาย Sequence Diagram โดยที่ทั้ง Use Case และ Sequence

Diagram ดังกล่าวนั้นจะต้องมีความเกี่ยวข้องกับ Class ตัวนั้น

รูปที่ 10 กระบวนการทำงานของ UML



หนังสืออ้างอิง

[1] *VB6 UML Design and Development*, Jake Sturm ,Wrox Press ltd. , 1999

[2] *Instant UML*, Pierre-Alain Muller , Wrox press Ltd , 1997

[3] *Applying Use Cases* , Geri Scheider , Jason P.Winters , Asddison-Wesley ,March 1999

ประวัติผู้เขียน

ดร.บรรจง หะรังษี เกิดที่จังหวัดลพบุรีเมื่อวันที่ 19 กรกฎาคม พ.ศ. 2508 จบการศึกษาระดับปริญญาตรี สาขาวิทยาการคอมพิวเตอร์ เกียรตินิยมอันดับสอง จากมหาวิทยาลัยหอการค้าไทยเมื่อ พ.ศ 2532 จบการศึกษาระดับปริญญาโท และปริญญาเอก สาขา Computer Science and Engineering จาก University of New South Wales, Australia เมื่อปี พ.ศ. 2537 และพ.ศ 2541 ตามลำดับ ได้เริ่มทำงานที่ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติในตำแหน่งนักวิจัย เมื่อเดือนมิถุนายน พ.ศ 2541 ปัจจุบันรับผิดชอบในงานด้านการวางแผน วิเคราะห์ และออกแบบโครงสร้างพื้นฐานของ NECTEC's MIS

นางญาณวรรณ สินธุภิญโญ จบการศึกษาระดับปริญญาโทจากจุฬาลงกรณ์มหาวิทยาลัย สาขาเทคโนโลยีสารสนเทศ เมื่อปี พศ.2538 เริ่มทำงานกับศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติในตำแหน่งนักวิจัย โดยรับผิดชอบงานในด้านการวิเคราะห์ ออกแบบและพัฒนาระบบสารสนเทศด้านต่างๆขององค์กร ในระหว่างเดือนตุลาคม พศ.2540 – พฤศจิกายน 2540 ได้รับทุนจาก the Japan International Co-operation Agency (JICA)เพื่อไปอบรมในด้าน Computer Software Technology ที่ประเทศสิงค์โปร์ ปัจจุบันปฏิบัติงานในงานโครงการเครือข่าย NECTECNet

Standardization Activities and Open Source Movements in Thailand[1]

Theppitak Karoonboonyanan, Thaweesak Koanantakool

National Electronics and Computer Technology Center
National Science and Technology Development Agency
Ministry of Science Technology and Environment, Thailand.
theppitak@nectec.or.th, htk@nectec.or.th

ABSTRACT — Standardization plays an important role in settling interoperability problems. This paper presents a survey on activities relating to IT standardization during the recent years in Thailand. Character sets, internationalization, font metrics, and studies on Tai scripts are discussed. In addition, the open source movements provide the stage for the standards to be realized and are beneficial to the promotion of the standards, as summarized in the paper.

บทคัดย่อ – การกำหนดมาตรฐานมีบทบาทสำคัญต่อการทำงานเข้ากันได้ของระบบต่างๆ บทความนี้นำเสนอบทสำรวจกิจกรรมต่างๆ ที่เกี่ยวข้องกับการกำหนดมาตรฐานเทคโนโลยีสารสนเทศในประเทศไทย อันประกอบด้วย รหัสอักขระ การทำให้เป็นสากล (internationalization) สัดส่วนขนาดของฟอนต์ภาษาไทย และการศึกษาตัวอักษรตระกูลไท นอกจากนี้ ยังกล่าวถึงความเคลื่อนไหวของซอฟต์แวร์ต้นรหัสเปิด (open source software) ซึ่งสามารถเป็นเวทีสำหรับการสร้างซอฟต์แวร์ที่ตรงตามมาตรฐาน และในขณะเดียวกัน ก็มีผลต่อการรณรงค์การใช้มาตรฐานเองอีกด้วย

0. Introduction

Standardization of IT in Thailand was recognized since 1984, when there were more than 26 sets of character codes were in use [1]. Two years later, an agreed standard code for Thai language was announced as a Thai Industrial Standard, TIS 620-2529/1986. However, at that time, only the codes were standardized. The input/output systems for computer processing [2] have not yet been unified. Operating systems and applications have been localized individually, based on different conventions. The proprietary standard that gains the lion's share in the market becomes *de facto*, no matter how its enhancement makes it deviated from industrial standards. Interoperability problem is therefore inevitable, especially in the age that different systems are connected through the Internet. Hence, standardization plays an important role in moderating the plethora of practices.

Recently, the open source paradigm has been widespread, and has become another model for

1 Published in the Proceedings of MLIT-4 : Fourth International Symposium on Standardization of Multilingual Information Technology, October 27-28, 1999, Yangon, Union of Myanmar. It is sponsored by CICC/MITI/MCF. Contact person of CICC is Mr. Takayuki Sato (sato@net.cicc.or.jp).

software development. The openness of the source code also gives the chance to control the conformance to the standards of the software, as well as the satisfaction to users' needs.

To shape consistent language support technology in the country, standardization activities and responses to open source movements are thus indispensable, and will be described in this paper.

1. Character Sets

The national standard character set for use in computers is TIS 620-2533/1990, from which several character sets are derived, for example, IBM code page 874 (cp-874), Microsoft code page 874 (windows-874) and Apple Thai (MacThai) [3]. These character sets are widely adopted in proprietary software, causing conflicts in communication among different platforms in the Internet.

Ironically, it's the game of the name. Only TIS 620 common characters are exchanged in practice, with different code set labels. The response to the code set with "unknown" name depends on applications. Some ignore the code set and process the text with their default preferences, while others simply reject.

Ad hoc solutions are also ubiquitous, such as using "iso-8859-1" or "x-user-defined" code name for Thai E-mails and web sites, by which Thai message could pass through the hole to the receiver in some weak situations. But that is not always the case.

In September 1998, the "tis-620" MIME character set has been registered by Trin Tantsetthi [4] with the Internet Assigned Number Authority (IANA) of the Internet Engineering Taskforce (IETF). A campaign has been set up by a group of developers [4] [5] to promote the use of the new standard MIME character set.

In 1999, the international standard ISO/IEC 8859-11 Latin/Thai characters has been reactivated by the ISO/IEC JTC1/SC2/WG2, and is becoming another potential choice of the standard MIME character set. When applied, "tis-620" and "iso-8859-11" are likely to be aliases to each other.

For multilingual documents, "utf-8" [6] is another possible alternative encoding. Nonetheless, the lack of UTF-8 editor is still the problem.

2. Internationalization

The third edition of ISO/IEC 14651 International String Ordering [7] has included an informative annex describing Thai string ordering. And, hopefully, the ordering of Thai in the standard would be satisfactory for Thai users.

A principle for Thai string ordering in detail has been proposed by a group of developers [8], and, as a consequence, the LC_COLLATE category of POSIX locale has been defined, as well as the other categories in a later time [9].

With the cooperation with the GNU C library project, the drafted POSIX locale has been made effective with glibc 2.1.1, which is used in modern distributions of Linux operating systems, such as Red Hat 6.0. Applications known to be internationalized and reflect the Thai locale include Linux 'date' and 'cal' commands, GNOME calendar, GNOME panel clock, KDE panel clock, and Perl 5.

3. Fonts

Thai fonts currently available in the market are designed based on Roman font metrics. This is not appropriate for Thai glyphs, since Thai characters are written in 4 levels. As a result, Thai glyphs are usually compressed to accommodate space for the 4 levels, and look smaller than Roman letters with the same point size.

The National Electronics and Computer Technology Center (NECTEC) therefore set up a committee for drafting the standard metrics for Thai glyphs relative to Roman and for creating prototype fonts to be used in public domain.

Three public-domain fonts, knowned as National Fonts (NF) 1, 2 and 3, are now available to the public. They are aimed to be the default fonts available in every platform. NF1 and NF3 are serif fonts. NF2 is sans serif. NF4 is planned for a "calligraphic" model font and NF5 is planned for a "handwriting" model font. Within December 1999, the official names of these fonts will be announced as part of the celebration of the 6th cycle anniversary (72nd birthday) of His Majesty The King of Thailand.

4. Tai Scripts Studies

Thai language used in the central Thailand belongs to the *Tai* language family. The scripts belonging to the family have caught the interests from a group of standardization committees. For example, New *Tai Lue* and *Tai Dam* scripts have been proposed to be encoded in the ISO/IEC 10646-1 character set.

In Thailand, Mr. Thawee Sawangpanyangkoon has done a research on *Tai* scripts and has created TrueType fonts for 13 *Tai* scripts, through the funding of the Thailand Research Fund (TRF).

We expect that more efforts will be made in the study of unification of these scripts with Thai scripts.

5. Open Source Movements

Several developers in Thailand have adopted the philosophy of open-source software in their works and have joined the world in this movement. Linux, the free OS of Linus Torvald, has become popular in Thailand and many developers have joined together in boosting the use of Thai language in the OS, with X Window as the GUI environment.

5.1 Distributions

There are currently four local Linux distributions in Thailand: Kaiwal Linux by Kaiwal Software, Linux School Internet Server (Linux SIS) and Linux with Thai Language Extension (Linux-TLE) by the National Electronics and Computer Technology Center (NECTEC), and Burapha Linux by Burapha University. These distribution developers meet regularly and join in regular Linux/Open-Source Symposia. It is expected that some distributions may merge in the new releases.

5.2 Development Projects

Several efforts are made to enable Thai language in open-source applications. Here are some examples:

1. **NACSIS-Thai Project** [10] is probably the first effort to support Thai on various platforms that are not Thai-localized.
2. **ZzzThai** [11] is another project to enable Thai in operating systems and applications on various platforms.
3. **Thai Linux Working Group** [12] is a Thai developer community concentrating on developments on Linux.
4. **WindowMaker** [13] is a GNU window manager project based on GNUStep. The Thai XKB with language mode locking allows user to input Thai characters conveniently. A Thai developer has also been one of the development team.
5. **Mozilla** [14] is an open-source project set up by Netscape Communication Co., Ltd., by opening the source code of its browser and other components. Three Thai developers have contributed the Thai language support to the browser [15]. Mozilla now can recognize the "tis-620" MIME character set, and can wrap Thai text lines appropriately.
6. **Thai X Terminal** is a free terminal emulator on X Window. It has been modified to enable Thai input/output for natural use.
7. **GNU Emacs** [16] now becomes multilingual. Collaboration between ETL and NECTEC has been set to add complete Thai language support and dictionary companion to the editor environment [17].
8. **MySQL** [18] database server has been modified to sort Thai fields appropriately [19].
9. **Thai POSIX Locale** [9] is a set of Thai cultural conventions for standard C library. It works with GNU LibC 2.1.1.
10. **Thai LaTeX,** based on Babel package, allows Thai documents preparation on Linux.
11. **Thai Library** is an effort to define standard API for Thai support in applications and to provide some chosen solutions.

6. Conclusion

Solutions and practices are usually one step further than the standards. In such situation, interoperability problem will call for new standards. The Internet has proven to be the main force in making new standard and

interoperability adopted a lot faster than in the past. More and more developers are now joining force in the making of standards and putting these standards to work

Open source model does not only provide a means of cooperative development, but also allows the software to be standardized, and the standard conventions to be realized. Therefore we take both streams as our means to develop our information technology for the future. We have illustrated the case of Thailand, which is now gaining a tremendous trust from the open-source movement. The outcome is amazing: something real, usable and stable enough for mission-critical applications.

References

[1] Thaweesak Koanantakool and the Thai API Consortium. **Computer and Thai Language.** National Electronics and Computer Technology Center, 1987. *ISBN* 974-7570-66-1. (in Thai)

[2] Thaweesak Koanantakool and Adshariya Agsorn-intara, **Character Codes and Input/Output Method for the Thai Language.** CICC/NACSIS/National Electronics and Computer Technology Center, Tokyo, Japan, 1990. *http://thaigate.nacsis.ac.jp/refer/thaiconf/*

[3] Trin Tantsetthi. **An Annotated Reference to the Thai Implementations.** *http://www.inet.co.th/cyberclub/trin/thairef/*

[4] Trin Tantsetthi. **Campaign for Internet-Standard-Conforming Thai Usage.** (in Thai) *http://software.thai.net/tis-620/*

[5] Worawit Khangtrakool. **TIS-620 Friendly Version.** (in Thai) *http://www.thai.net/tis-620/*

[6] F. Yergeau. **UTF-8, a transformation format of ISO 10646.** RFC 2279. Alis Technologies. January 1998. (Obsoletes RFC 2044)

[7] Alain LaBonté. (Editor) **ISO/IEC FCD 14651.3 – International String Ordering – Method for comparing Character Strings and Description of the Common Template Tailorable Ordering.** June 1999.

[8] Theppitak Karoonboonyanan, Samphan Raruenrom and Pruet Boonma. **Thai-English Bilingual Sorting.** *http://www.links.nectec.or.th/~thep/blsort.html*

[9] Theppitak Karoonboonyanan. **Thai Locale.** *http://www.links.nectec.or.th/~thep/th-locale/*

[10] National Center for Science Information Systems. **NACSIS-Thai Project.** *http://thaigate.rd.nacsis.ac.jp/*

[11] uecthai@fedu.uec.ac.jp. **ZzzTh@i : How to use Thai with various computer platforms.** *http://zzzthai.fedu.uec.ac.jp/*

[12] Thai Linux Working Group. **Thai Linux Working Group.** *http://linux.thai.net/*

[13] Alfredo K. Kojima. **WindowMaker.** *http://www.windowmaker.org/*

[14] The Mozilla Organization. **Mozilla.org.** *http://www.mozilla.org/*

[15] Samphan Raruenrom. **Mozilla Thai Enabling.** *http://developer.thai.net/mozilla/*

[16] Free Software Foundation. **GNU Emacs.** *http://www.gnu.org/software/emacs/emacs.html*

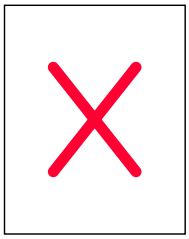
[17] Software and Language Engineering Laboratory, NECTEC. **Thai Language Support for Emacs.** *http://www.links.nectec.or.th/themacs/*

[18] T.c.X DataKonsultAB. **MySQL by T.c.X DataKonsultAB.** *http://www.mysql.net/*

[19] Pruet Boonma. **Thai Sorting Support for Free Database Server.** *http://linux.intanon.nectec.or.th/thaisortdatabase.html*

BIOGRAPHIES

Theppitak Karoonboonyanan

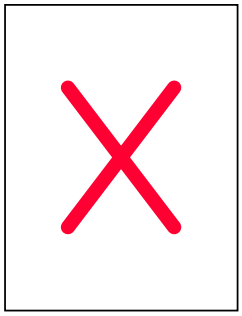


Theppitak Karoonboonyanan received his B.Eng. (Computer Engineering 2nd class honor) from

Chulalongkorn University in 1993. He joined the Software and Language Engineering Laboratory (SLL), NECTEC since August 1995 as a research assistant. His experience includes Thai word processor and the Royal Institute Dictionary CD-ROM. This made him interested in Thai algorithms and implementations. He has also been involved in a number of Thai open-source projects, such as Thai POSIX locale. His interests include object-oriented software engineering and artificial intelligence.

Dr. Thaweesak Koanantakool

Director of NECTEC



Dr. Thaweesak "Hugh" Koanantakool received his Bachelor and Ph.D. degrees in Electrical Engineering from Imperial College of Science and Technology, London University. He had a number of industrial contracts in the UK before he came back to Thailand to start his government service career in 1981. He taught in Electrical Engineering with the Faculty of Engineering, Prince of Songkla University. In 1985, he moved to Bangkok, Thammasat University, and was appointed Associate Director of the Information Processing Institute for Education and Development. Since 1994, he became Deputy Director of NECTEC as well as leading the Network/Software Technology labs. Thaweesak introduced the Internet into Thailand and set up the largest academic and research network known as ThaiSarn under NECTEC. He later co-founded the first Internet Service Provider (ISP) owned by Thai government in 1995. The ISP, Internet Thailand Company Limited, at present is the largest ISP in Thailand, has 45% market share (by IP numbers managed). In 1996-1997, Thaweesak led Thailand's Information Superhighway test bed Project funded by NECTEC. The project was a major test bed in Thailand using ATM switches for both local area and wide-areas. In August 1998, Thaweesak was appointed the Director of NECTEC.

LETTERS

Dear all,

On behalf of the editor-in-chief of NTJ, I would like to express my appreciation to all authors who submitted papers, reviewers, and who all have contributed to the success of NTJ up to this point. I am very glad to inform you that, during this period of time, there will be two well-known conferences taking place in Thailand. One is a 1999 IEEE International Symposium on Intelligence Signal Processing and Communication Systems (ISPACS'99) and the other is the 1999 National Computer Science and Engineering Conference (NCSEC'99).

ISPACS'99

ISPACS'99 will be held during December 8-10, 1999 at Phuket Arcadia Hotel & Resort. In the symposium, there will be about 200 papers for presentation in 4 parallel tracks of 24 lecture sessions and 1 track of 6 poster sessions, with 2 special sessions on Thai language processing and VLSI architecture.

There are two tutorial sessions : ***"MPEG-4"*** by K.R. Rao from university of Texas at Arlington, ***"Wireless Telecommunication Techniques for Fading and Interference Environments"*** by Takis Mathiopoulos from university of British Columbia. Futhermore, there are four keynote speeches : ***"Challenges in Realizing the Multimedia Mobile Communications Era : IMT-2000 and Beyond"*** by Fumiyuki Adachi from NTT Mobile Communications Network, Inc., ***"A Dynamic Networking Architecture Networks for the Next Generation"*** by Norio Shiratori from Tohoku University, ***"New Challenge of VLSI Design in System-on-Silicon Era"*** by Hiroaki Kunieda from Tokyo Institute of Technology, and ***"Intelligent Signal Processing in Wireless Systems"*** by H. Vincent Poor from Princeton university.

This symposium is sponsored by IEEE Communication Society, NSTDA, NECTEC, JICA, SIIT. For more information, please contact ISPACS'99 secretariat office at 662-7372500 ext. 5023, 5024, e-mail : ispacs99@kmitl.ac.th, and http://www.kmitl.ac.th/~reccit.

NCSEC'99

NCSEC'99 is the only national conference dedicated specifically for the fields of Computer Science and Computer Engineering. It will be held on December 16-17, 1999 at Landmark Hotel, Bangkok. Over 55 papers will be presented in 3 parallel tracks of 16 lecture sessions and 2 poster sessions. Two especially large sessions are on Thai Character Processing and on Intelligent Systems.

Besides the paper presentation, a panel discussion on the topic "Software Engineering in Computer Science and Computer Engineering Education" is specially conducted in order to reflex the national need for this particular technological development.

Four tutorial sessions consist of ***"High-Performance Numerical Linear Algebra: Fast and Robust Kernels for Scientific Computing"*** by Jack Dongarra from University of Tennessee at Knoxville and Oak Ridge National Laboratory, USA**,** ***"Bayesian Networks"*** by Peter Haddawy from Assumption University and University of Wisconsin at Milwaukee, ***"Genetic Algorithm and Its Applications"*** by Prabhas Jongsatitwattana from Chulalongkorn University, and ***"Education System Reform in the Next Millennium – A Case Study"*** by Yuen Poovarawan from Kasetsart University.

A special corner will also be dedicated to the exhibition and a series of brief presentations about ***"Linux Clustering"*** headed by Putchong Uthayopas from Kasetsart University.

On December 16, The keynote speech by Jack Dongarra will be on the topic ***"High-Performance Computing and Netsolve: A Network Server for Solving Computational Science Problems".*** On December 17, there will be an invited talk by Thaweesak

Koanantakool, director of the National Electronics and Computer Technology Center (NECTEC).

The NCSEC committee consists of a number of lecturers and researchers from many universities in Thailand. NCSEC'99 is hosted by Department of Computer Science, Faculty of Science and Technology, Assumption University, and is sponsored by Assumption University and NECTEC. For more information, please contact 300-4543 – 53 ext. 3680..2, e-mail : ncsec99@s-t.au.ac.th and http://www.s-t.au.ac.th/~ncsec99.

I hope these two conferences will stimulate basic researches in our beloved country to be full with a sense of well-being and will solidify the freindship among the computer scientists and engineers in the fields.

Furthermore, there is a letter from Assoc. Prof. Preecha Yupapin states fact about basic researches in Thailand in the field of optics. I hope it contains a valuable information for optics interested groups of people. Lastly, I would like to cordially invite you to submit papers, tutorials, and letters to our NECTEC Technical Journal to disseminate knowledge to the public.

Chularat Tanprasert

Editor-in-Chief

Research in Optics Technology

We are recognized that basic research in science and technology as much as product oriented development firmly economical growth. Optics is one of the major subjects which is popularly studied in the next decade.

Astonishing as the demand for wavelength division multiplexing, (WDM) has been rapidly growth for the use in WDM systems. Each category of components has experienced this rise, including lasers, multiplexers and demultiplexers, and amplifiers. The trend toward blending DWDM and data transport technologies is creating new opportunities for optical add-drop multiplexers and cross-connects.

Optics research in Thailand is increased in the next century due to the demand of technology and industry. Optical technology has shown the advantage such as the use in communication for high speed communications, optical computer, holographic movie, optical metrology and medical optics, etc., which may be divided into the following groups.

Optical Theory

This group studies the basic theory in optics concerning optical properties and related areas, for examples, physical optics, geometrical optics, holography and metrology. These areas are studied to support the experimental groups. Numerical method is also used for the prediction of the theoretical study in wide area of research.

Fiber Optics

Fiber optics with the applications in either communication or sensors are studied in this group, heading for principle of new devices and technologies such as optical switching, optical filter, optical signal processing, i.e WDM technology and the use of plastic optical fibers, nonlinear properties such as optical bistability or chaotic switches, soliton and fiber amplifier.

Photonic Materials

The searching of new materials for photonic devices are focused in this group. These devices are concerned about the material applications for transmitters, receivers and modulators, where the applications can be more useful and helpful if the device characteristics can be described concerning the realistic applications.

Laser Applications

Department of Applied Physics,
Faculty of Science, KMITL

This group uses laser to study the optical material in the form of thin film which are concerned their optical properties,. The material processing by laser is also investigated, where the application linking to the industry is the major activities of this group. Laser theory is also studied in this group. The designing of laser system, laser cavity and components are also in the plan of their research.

Optoelectronic Devices

They work on quantum well devices for transmission and detection, either for sensors or communications which is widely explored based on semiconductor and quantum well devices. The devices characteristics are also studied with networking control.

Acousto-Optics

This group is focussed on the use of acousto-optic modulator for optical communication, i.e. external moderation either experimental or theretical works.

Magneto-Optics

They work are the study of optical properties of magnetic material which is focussed on the technique to serve either basic research and applications in science and technology.

KMITL is known as a leadership institution in applied science and technology, where the basic optical research and technology has been taken place in either science and engineering faculties. There are 8 research and development (R&D) groups with a total of 50 researchers working in optical technology and related fields. There is a Ph.D. program by research which has been established in the Department of Applied Physics, Faculty of Science in either national or international programs.

Assoc.Prof. Dr. Preecha Yupapin
Lightwave Technology Research Center,