국제회의 기고서

ITU-T SG 회의 국내 기고서

내 역

본란은 스위스 제네바에서 개최된 ITU-T SG10회의 ('93. 10. 19~10. 28) 와 ITU-T SG8회의 ('93. 11. 16~11. 25) 에 참석한 국가 참가단에서 발표한 지연기고서 (Delayed Contribution) 내용을 소개하는 것입니다.

UIT—Secteur de la normalisation des telecommunications
ITU—Telecommunication Standardization Sector
UIT—Sector de Normalization de las Telecommunications

Commission d'etudes)
Study Group) 10
Commision de Estudio)

Contribution tardive)
Delayed Contribution) D.
Contribution tardia)

Geneve, 19—28 October 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 10

SOURCE : KOREA(REPUBLIC OF)

TITLE : 객체지향 CHILL을 위한 중복 기능

요 약

본 기고서는 객체지향 CHILL을 설계하기 위한 하나의 작업으로써 중복 기능의 다양한 측면들을 간략하게 나타내고자 한다.

1. 서론

일반적으로 프로그래밍 언어에서, 특히 Ada, 중복 기능이 순수한 객체지향 기능들에 속하진 않지만 사용자를 위해 구문의 표현력 증대를 제공한다. 다른 한편으론 [CW85]의 정의에 의하면 중복 기능이(객체지향 개념인)다향성 중의 한 부분으로 간주되기도 한다. 왜냐하면 하나의 이름이 같은 범주내에서 여러 다른 행위를 가질 수 있기 때문이다. 이러한 중복 개념은(이름 혹은 연산자 프로시듀어, 리터럴, 특히 set 원소들 처럼)같은 이름으로 여러 발생하는 것들이 중복에서 허용된다. 즉 하나의 발생자가 같은 범주내에서 동시에 각기 다른 의미들을 갖는다.

CHILL에서는 프로시듀어의 이름들이(MODULE 처럼) 같은 범주내에서 유일해야하고 연산자 기호가 프로시듀어의 이름으로 될 수 없다는 제약들을 갖는다. 또한 리터럴 중복은 프로시듀어들의 연산자 기호와 이름들이 중복되는 것과 달리 set 원소들의 중복 사용을 허용함으로써 프로그래머에게 애매모호성을 가져다 준다. 그리고 프로시듀어의 이름들이 추상데이타타입(ADT)밖에서 중복으로 허용될때, 이의 효용가치가 의심스럽다. 왜냐하면 새로 정의한 프로시듀어 이름의 중복과 이미 정의된 프로시듀어 이름들간에 혼동을 유발할 가능성이 있다.

작성자 : 이준경 선임연구원, 최완 선임연구원(ETRI 개발환경연구실)

2. 중복의 종류

Ada[Ic91] 언어의 중복 기능에 맞추어 분류하였다. 중복 개념을 일반적인 문장에 적용하면 다음과 같다. "He got home."이라는 문장에서 동사인 "got"가 여러 뜻을 가져도 주어나 목적어인 명사 등에 의해 하나의 의미를 구별할 수 있다. 만일 명사가 유일하지 못하면, 동사의 뜻이 문맥이나 (C++의 범주해결연산자 같은) 첨자 방식에 의해 구별할 수 있다. 그러나 이는 구문을 복잡하게 하고 컴파일러의 구현을 복잡하게 만든다.

연산자 및 이름의 중복 프로시듀어들의 정적 바인딩은 형식-매개인자들의 모드, 갯수, 순서 그리고 복귀-타입들에 대응된다. 그러므로 다음의 CHILL 프로그램을 위주로 각 중복 기능들의 자세한 사항들을 나타내고자 한다.

```
SYNMODE st=STRUCT(x INT;
                  y INT);
SYNMODE ar=ARRAY(1:2) INT;
DCL a st:=[10, 20],
   b st:=[100, 200],
   c st;
   d ar:=[10, 20],
   e ar:=[100, 200],
   f ar:
addS:
PROC(p1 st, p2 st) RETURNS(st);
   DCL t st:=[p1.x+p2.x, p1.y+p2.y];
   RETURN(t);
END addS;
addA:
PROC(p1 ar, p2 ar) RETURNS(ar);
   DCL t ar:=[p1(1)+p2(1), p1(2)+p2(2)];
   RETURN(t);
END addA;
c:=addS(a, b);/*구조체의 덧셈 */
f:=addA(d, e);/*배열의 덧셈 */
```

2.1 연산자 중복

다른 모드와 다른 행위를 갖는 많은 프로시듀어들이 내부 연산의 같은 의미를 가질때 연산자 중복은 수식 구문의 표현에 용이함을 준다. 이러한 기능은 사용자의 추상적 파악에 도움을 주고 구문의 함축성(예:add(add(a, b), c)=>a+b+c)을 제공한다. 다음 예제는 연산자 중복에 관하여 보

여준다.

```
"+":
PROC(p1 st, p2 st) RETURNS(st);
   DCL t st:=[p1.x+p2.x, p1.y+p2.y];
   RETURN(t);
END "+";
"+":
PROC(p1 ar, p2 ar) RETURNS(ar);
   DCL t ar:=[p1(1)+p2(1), p1(2)+p2(2)];
   RETURN(t);
END "+";
c:=a+b;/*구조체의 덧셈에 대한 간략한 구문 */
f:=d+e;/*배열의 덧셈에 대한 간략한 구문 */
```

중복 프로시듀어에 적용될 수 있는 연산자들의 우선순위를 변경할 수 없지만 수식구문은 새로이 만들 수 있다. 예를 들면, i++와 ++i는 i의 값을 1만큼 증가/감소시키는 것을 의미한다. 그러나 새로운 수식 구문들을 허용함으로써 연산자 우선순위를 적용하기 위해서 이의 허용에 대하여 토의해야 한다. 따라서 연산자 중복은 유용한 기능이지만 C++[St86]같이 많은 기능들 중에서 몇개의 연산자만을 허용하는 제약을 가해야 한다. 다음은 중복 기능을 객체지향 CHILL에 도입하는데 적합할 연산자들을 나열하였다.

1) 단항/이항 연산자(+, −. *, /, mod, rem)
   단항 연산자 프로시듀어의 매개인자 갯수는 1개이고, 다항은 2개여야 한다. 매개인자 전달은 모두 IN이어야 한다. 이에 적용되는 피연산들의 모드는 원시 모드가 자동적일 뿐만아니라 사용자-정의 모드들에 대해 허용된다. 이 연산자들의 우선순위는 변경되지 말아야 한다.
2) 논리 연산자(and, or, not)
   매개인자 갯수는 2개이고, 전달 방식은 모두 IN이어야 한다. 복귀 모드는 boolean어어야 한다.
3) 비교 연산자(<, <=, =, /=, >=, >)
   매개인자 갯수는 2개이고, 전달 방식은 모두 IN이어야 한다. 복귀 모드는 boolean어어야 한다.
4) 치환 연산자(:=)
   매개인자 갯수는 2개이고, 전달 방식은 LOC과 IN이어야 한다.

2.2 이름 중복

객체지향 CHILL은 기존의 int/float 모드간의 연산들을 제외한 변수나 상수의 새로운 중복 기능을 허용하지 말아야 한다. 이의 허용은 "DCL o colorType, o fruitType;"같은 잠재적인 애매모호성을 갖는다. 그래서 이름 중복은 오직 프로시듀어만을 허용해야 한다. 다음 예제는 앞에서 언급한 "+"연산자 중복에 대해 이름 프로시듀어로 재정의했다.

```
add:
PROC(p1 st, p2 st) RETURNS(st);
   DCL t st:=[p1.x+p2.x, p1.y+p2.y];
   RETURN(t);
END add;
add:
PROC(p1 ar, p2 ar) RETURNS(ar);
   DCL t ar:=[p1(1)+p2(1), p1(2)+p2(2)];
   RETURN(t);
END add;
c:=add(a, b);/* 구조체의 덧셈 */
f:=add(d, e);/* 배열의 덧셈 */
```

2.3 리터럴 중복

리터럴은 상수의 일종으로, ①다른 값을 갖지만 같은 기호를 갖는 set 원소나 ② (int, float, bool, char, null, string)유일한 계산 값들로 구성된다. 여기서 ①기능은 CCITT에서 합의한 "약가시성 제거"에 의해 언급할 필요가 없다. 그러나 ②기능은 기존의 CHILL에서 허용하고 있다. 이에 대한 예를 살펴보면 다음과 같다.

```
SYNMODE colorType=SET(red, orange, yellow, green),
            fruitType=SET(apple, banana, orange, kiwi);
DCL color colorType:=orange, /* orange=1 */
      fruit fruitType:=orange; /* orange=2 */
DCL x INT:=1.0, /* 내부적으로 정수 1로 변환 */
      y FLOAT:=1; /* 내부적으로 float 1.0으로 변환 */
```

참 고 문 헌

[CW85] L.Cardelli, P.Wegner : On Understanding Types, Data Abstraction, and Polymorphism : Computing Surveys, 17, 4(1985) 471−522

[Ic91] J.Ichbiah, J.Barnes, R.Firth, M.Woodger : Rationale for the Design of the Ada Programming Language : Cambridge Univ. Press 1991

[St91] B.Stroustrup : The C++ Programming Language : Addison−Wesley, Reading, Mass., second edition, 1991

[TD405−E] WP Chairman : Removal of weak visibility : Temporary Document 405−E 1991

UIT - Secteur de la normalisation des telecommunications
ITU - Telecommunication Standardization Sector
UIT - Sector de Normalization de las Telecommunications

Commission d'etudes)
Study Group) 10
Commision de Estudio)

Contribution tardive)
Delayed Contribution) D.
Contribution tardia)

Geneve, 19-28 October 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 10

SOURCE : KOREA (REPUBLIC OF)

TITLE : Overloading Features for Object-Oriented CHILL

Abstract

This contribution presents briefly the aspects of overloading features for the design of Object-Oriented CHILL.

1. Introduction

Generally, even if it is considered that the overloading feature in programming languages, especially Ada, is not a part of pure object-oriented features, the overloading feature supports the facility for the descriptive power of programmers. On the other hand, the overloading is considered as a part of polymorphism by the definition of [CW85]. Because a name may have many different behaviors in the same scope. This means that the multiple occurrences with the same name (such as procedures prefixed by names or operators, literal constants including set elements) are allowed for overloading. In overloading, an occurrence has the different meanings simultaneously in the same scope.

CHILL has the constraints that the names of procedures are unique in the same scope (like in MODULE) and the operator symbols are not allowed to be procedure names. Since literal overloading may cause the serious problems for allowing the multiple occurrences of set elements, it leads a potential ambiguity for programmers. And then we doubt that it is beneficial to the permission for procedure names to be overloaded outside the abstract data types because it may make the understandability of OO-CHILL rather complex and may lead to the ambiguity between the newly defined procedure names and the pre-defined procedure names.

2. Kinds of Overloading

These three kinds are classified by Ada's approach[Ic91]. When the overloading concept is applied to the general sentence as naturally, it can be presented as follows. In the sentence, such as "He got home.", the verb of "got" may have many different meanings but can be distinguished by the information of its subject nouns, its objective nouns and the information of relative contexts. That is, if the overloaded symbols may be used, the meanings of the symbols can be distinguished by its contexts or the use of qualification (similar to C++'s scope resolution operator). But its syntax and semantics would be rather complex, and it makes hard to implement.

The static bindings of the operators' procedures and the names' procedures for overloading should be corresponded to the modes, the number and the ordering of the formal parameters, and the mode of the return-spec. Therefore, so as to show the examples for the overloadings of operators and names procedures, and literals, we try to present the detailed considerations from the following CHILL program.

```
SYNMODE   st = STRUCT(x INT;
                      y INT);
SYNMODE   ar = ARRAY(1:2) INT;
DCL   a st := [10,20],
      b st := [100,200],
      c st,
      d ar := [10,20],
      e ar := [100,200],
      f ar;

addS:
PROC(p1 st, p2 st) RETURNS(st);
     DCL   t st := [p1.x+p2.x, p1.y+p2.y];
     RETURN(t);
END addS;

addA:
PROC(p1 ar, p2 ar) RETURNS(ar);
     DCL   t ar := [p1(1)+p2(1), p1(2)+p2(2)];
     RETURN(t);
END addA;

c := addS(a,b);   /* add to structure values */
f := addA(d,e);   /* add to array values */
```

2.1 Overloading of Operators

When many overloaded procedures with the different modes and behaviors have the same meanings about the internal operations, the overloading of operators can

be used for the expression styles. These features help the abstract understandability by the programmers and support the abbreviations for the sugar syntax (eg. add(add (a,b),c) => a+b+c). The following example shows about the overloading of operators.

```
"+":
PROC(p1 st, p2 st) RETURNS(st);
      DCL   t st := [p1.x+p2.x, p1.y+p2.y];
      RETURN(t);
END "+";

"+":
PROC(p1 ar, p2 ar) RETURNS(ar);
      DCL   t ar := [p1(1)+p2(1), p1(2)+p2(2)];
      RETURN(t);
END "+";

c := a + b;   /* sugar syntax for structure modes */
f := d + e;   /* sugar syntax for array modes */
```

It should not be allowed to change the precedence for the overloading of operators due to its ambiguity. It may be allowed to make new syntax of expressions. For example, "i++" or "++i" can be allowed in that case. But we should discuss the permission of the overloading of newly defined operators because its new precedence would be added to the existing precedence. We propose that these features, such as "++" etc, would not be included in the reserved symbols. Finally we think that the overloading of operators is necessary for the users' facilities. But the only few operators, which be classified by the followings of applicable CHILL symbols corresponding to the various features of operator overloading in C++[St91], should be allowed for OO-CHILL.

1) unary/binary operators (+, -, *, /, mod, rem)
   The number of parameters in the unary and the binary operators should be one and two. The attributes of parameter passing must be all of IN. Its operands are allowed for the user-defined modes as well as the primitive modes by default. Its precedence of the unary and binary operations must be unchanged.

2) logical operator (and, or, not)
   The number of parameters must be two, and the parameter attributes are all of IN. Its result mode must be boolean.

3) relational operator (=,/=,>,>=,<,<=)
   The number of parameters must be two, and the parameter attributes are all of IN. Its result mode must be boolean.

4) assignment operator (:=)

The number of parameters must be two, and one of the parameter's attributes should be LOC and the other's IN.

2.2 Overloading of Names

OO-CHILL should not permit the overloading of variables and constants except the existing int/float modes and so on. The permission causes a potential ambiguity, such as "DCL o colorType, o fruitType;". So the overloading of names is constrained to the only procedures. For the above overloading of "+" operator, the following example is shown as the overloading of procedure names.

```
add:
PROC(p1 st, p2 st) RETURNS(st);
     DCL  t st := [p1.x+p2.x, p1.y+p2.y];
     RETURN(t);
END add;

add:
PROC(p1 ar, p2 ar) RETURNS(ar);
     DCL  t ar := [p1(1)+p2(1), p1(2)+p2(2)];
     RETURN(t);
END add;

c := add(a,b);  /* the same name for structure mode */
f := add(d,e);  /* the same name for array mode */
```

2.3 Overloading of Literals

Literals stand for values, and can be classified by (1) set elements of same names with different values, of different modes, (2) numeric values (int, float, bool, char, null and string) with same values. It is not necessary to discuss the issue (1) because of the compromised issue about the "Removal of Weak Visibility" [TD405-E] in the previous meeting of '89-'92 period. The issue (2) is now allowed in CHILL.

```
SYNMODE   colorType = SET(red, orange, yellow, green),
          fruitType = SET(apple, banana, orange, kiwi);

DCL   color colorType := orange,    /* orange = 1 */
      fruit fruitType := orange;    /* orange = 2 */
DCL   x INT := 1.0,    /* convert to int. mode implicitly */
      y FLOAT := 1;    /* convert to float mode implicitly */
```

References

[CW85] L.Cardelli, P.Wegner : On Understanding Types, Data Abstraction, and Polymorphism : Computing Surveys, 17, 4 (1985) 471-522

[Ic91] J. Ichbiah, J. Barnes, R. Firth, M. Woodger : Rationale for the Design of the Ada Programming Language : Cambridge Univ. Press 1991

[St91] B. Stroustrup : The C++ Programming Language : Addison-Wesley, Reading, Mass., second edition, 1991

[TD405-E] WP Chairman : Removal of Weak Visibility : Temporary Document 405-E 1991

UIT—Secteur de la normalisation des telecommunications
ITU—Telecommunication Standardization Sector
UIT—Sector de Normalization de las Telecommunications

Commission d'etudes)
Study Group) 10
Commision de Estudio)

Contribution tardive)
Delayed Contribution) D.
Contribution tardia)

Geneve, 19—28 October 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 10

SOURCE : KOREA(REPUBLIC OF)

TITLE : 객체지향 CHILL을 위한 지연 프로시듀어

요 약

본 기고서는 클래스 계층구조에서 지연 프로시듀어를 나타내는 방식에 대해 언어 설계상의 고려사항을 나타내고자 한다.

1. 서론

```
class A {…
  public:
    void P(A *x) {…}
  };
class B:A {…
  public:
    void P(B *x) {…}
  };
f(){
A a;
B b,
  *rb=&b;
rb->p(&a); /* A::P()가 아닌 B::P()를 호출, 오류 발생 */
           /* 해결방안: rb->A::P(&a) */
}
```

작성자 : 이준경 선임연구원, 최완 선임연구원(ETRI 개발환경연구실)

우리의 제안은 위의 예제처럼 다음 가정을 둔다. 상위 클래스에서 정의한 프로시듀어가 같은 이름으로 하위 클래스에서 재정의 되었을때 범주해결연산자 없이는 하위 클래스에서 생성된 객체는 상위 클래스의 프로시듀어를 호출할 수 없다 [TD406-E]. C++의 가상함수에 대한 예제 프로그램은 다음과 같다.

```
  class A {…
    public: virtual void Print() {}
    };
  class B : public A {…
    public: void Print() {…}
    };
  class C : public A {…
    public: void Print() {…}
    };
  f() {
A a;
B b;
C c;
A *ra;
  ra=&a;
  ra->Print(); /* A::Print() 호출 */
  ra=&b;
  ra->Print(); /* B::Print() 호출 */
  ra=&c;
  ra->Print(); /* C::Print() 호출 */
  }
```

base 클래스에 있는 지연 프로시듀어 [TD319-E]를 C++에서는 "virtual"이라는 예약어를 이용하여 추상적으로 선언한다. 실제 예제 [St91]에서, 하위 클래스는 상위 클래스의 프로시듀어를 물려받지만 하위 클래스에서 재정의한 프로시듀어에게 은닉이 된다. 즉, 이러한 개념으로 바라볼때 상위 클래스는 선언적 의미를 가지고 하위 클래스는 구현적 의미를 가지는 것으로 간주하고 있다.

2. 제안

C++의 구현사항 [St87]을 살펴보면, 물려받는 상위 클래스의 프로시듀어를 하위 클래스에서 재정의한 프로시듀어로 치환됨을 알 수 있다. 이러한 프로시듀어 치환의 의미를 Modula-3 [Ca92]에서는 "overrides"라는 예약어를 이용하여 하위 클래스의 치환할 프로시듀어들 지역앞에 선언함

으로써 프로시듀어 재정의를 나타내고 있다. 따라서 객체지향 CHILL에서도 개념적 표현인 상위 클래스의 해당 프로시듀어에 표기하는 것(C++ 방식)보다, 구현적 표현인 하위 클래스의 해당 프로시듀어에 표기하는 것(Modula-3 방식)이 더 좋다고 생각한다. 왜냐하면 사용자가 클래스 계층구조상에서 어느 클래스의 프로시듀어들이 대치되는 가를 쉽게 파악할 수 있기 때문이다. 그러나 본 제안에는 지연 프로시듀어를 어떻게 표기할 지에 대해서는 여러가지 방안들이 있을수 있으므로 언급하지 않겠다.

다음 예제 프로그램은 "A <- B <- C"구조의 클래스 계층구조를 갖는다. 여기서 (1)은 C++처럼 상위 클래스에 표기하는 것이고, (2)는 Modula-3처럼 하위 클래스에 표기하는 것을 나타낸다.

```
SYNMODE A=OBJECT
  P:PROC(…)…END P; <--(1):"virtual"
END A;
SYNMODE B=OBJECT OF A
  P:PROC(…)…END P; <--(2):"overrides"
END B;
SYNMODE C=OBJECT OF B
  P:PROC(…)…END P; <--(2):"overrides"
END B;
```

참 고 문 헌

[Ca92] L.Cardelli, J.Donahue, L.Glassman, M.Jordan, B.Kalsow, G.Nelson:Modula-3 Language Definition:ACM SIGPLAN Notices, Vol.27, No.8 August 1992 15-42

[St87] B.Stroustrup:Multiple Inheritance for C++:Proceedings EUUG Mass., 1986

[St91] B.Stroustrup:The C++ Programming Language:Addison-Wesley, Reading, Mass., second edition, 1991

[TD319-E] WP Chairman:OOP in CHILL:Temporary Document 319-E Febrary 1991

[TD406-E] Telettra:Object-oriented CHILL:Temporary Document 406-E April 1990

UIT - Secteur de la normalisation des telecommunications
ITU - Telecommunication Standardization Sector
UIT - Sector de Normalization de las Telecommunications

Commission d'etudes) Contribution tardive)
Study Group) 10 Delayed Contribution) D.
Commision de Estudio) Contribution tardia)

Geneve, 19-28 October 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 10

SOURCE : KOREA (REPUBLIC OF)

TITLE : Deferred Procedures for Object-Oriented CHILL

Abstract

This contribution presents the consideration of deferred procedures for object-oriented CHILL.

1. Introduction

```
class A { ...
      public:   void P(A *x) { ... }
      };

class B : public A { ...
      public:   void P(B *x) { ... }
      };

f() {
A   a;
B   b,
    *rb = &b;
rb->P(&a);   /* call to B::P(), not A::P(), illegal use */
             /* resolved by rb->A::P(&a) */
}
```

Our proposal is based on the assumption as shown the above example that when a procedure defined in a derived class has the same name as that defined in a base class, the object created by the derived class cannot access the procedure with same name in the base class without a scope resolution operator[TD406-E]. The following shows an example of C++ virtual functions.

```
class A { ...
      public:   virtual void Print() { }
      };
```

```
class B : public A { ...
      public:   void Print() { ... }
      };

class C : public A { ...
      public:   void Print() { ... }
      };

f() {
A   a;
B   b;
C   c;
A   *ra;

ra = &a;
ra->Print();      /* call A::Print() */
ra = &b;
ra->Print();      /* call B::Print() */
ra = &c;
ra->Print();      /* call C::Print() */
}
```

The deferred procedure[TD319-E] in a base class is declared abstractly with "deferred" keyword in OO-CHILL (as "virtual" keyword used in C++). In its detailed implementation[St87] about deferred procedures, the object created by derived class has the newly defined procedures in its own class, and hides the inherited procedure from its base class. Thus, for the deferred procedures in C++, a base class has a meaning for declaration, and a derived class has a meaning for implementation.

2. Proposal

When we investigate the implementation of virtual functions in C++[St91], however, we found the machanism that the derived object does not inherit the deferred procedure declared in a base class but has the corresponding procedures defined with the same signature in the derived class. In Modula-3 [Ca92], procedure redefinition for deferred procedures is represented by using the "overrides" keyword declared in the derived class. Here we prefer the style of Modula-3 for the procedure redefinition to that style of C++. Because the users can find easily the position of procedure redefinition in the class hierarchy.

In our proposal, we want to treat only the marked position for deferred procedures in the class hierarchy, but not to its syntax and semantics. The following example presents the position for the class hierarchy with the "A <- B <- C". The following shows two approaches for the representation of procedure redefinition that the position-(1) may be marked by "virtual" ("deferred" or any syntax), or the position-(2) may be marked

by "overrides" (or any syntax).

```
SYNMODE   A = OBJECT
     P:PROC(...) ... END P;   <-- (1): "virtual"
  END A;

SYNMODE   B = OBJECT OF A
     P:PROC(...) ... END P;   <-- (2): "overrides"
  END B;

SYNMODE   C = OBJECT OF B
     P:PROC(...) ... END P;   <-- (2): "overrides"
  END B;
```

References

[Ca92] L.Cardelli, J.Donahue, L.Glassman, M.Jordan, B.Kalsow, G.Nelson : Modula-3 Language Definition : ACM SIGPLAN Notices, Vol.27, No.8 August 1992 15-42

[St87] B. Stroustrup : Multiple Inheritance for C++ : Proceedings EUUG Mass., 1987

[St91] B. Stroustrup : The C++ Programming Language : Addison-Wesley, Reading, Mass., second edition, 1991

[TD319-E] WP Chairman : OOP in CHILL : Temporary Document 319-E Febrary 1991

[TD406-E] Telettra : Object-oriented CHILL : Temporary Document 406-E April 1990

UIT—Secteur de la normalisation des telecommunications
ITU—Telecommunication Standardization Sector
UIT—Sector de Normalization de las Telecommunications

Commission d'etudes)
Study Group) 8
Commision de Estudio)

Contribution tardive)
Delayed Contribution) D.
Contribution tardia)

Geneve, 16—25 November 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 3, 15/8

SOURCE : KOREA(REPUBLIC OF)

TITLE : ODA문서 내용부(content portion)내에서 처리위치 지정을 위한 메카니즘의 필요성

ABSTRACT

ODA문서는 객체의 계층구조로서 구성되며, 각 객체는 객체식별자(object indentifier)를 통하여 문서내에서 지정된다. 문서를 구성하는 실제의 정보는 기본객체에 연결되는 내용부에 담겨지게 되는데, 현재 ODA에서는 내용부내의 특정위치를 지정하기 위한 메카니즘이 존재하지 않는다.

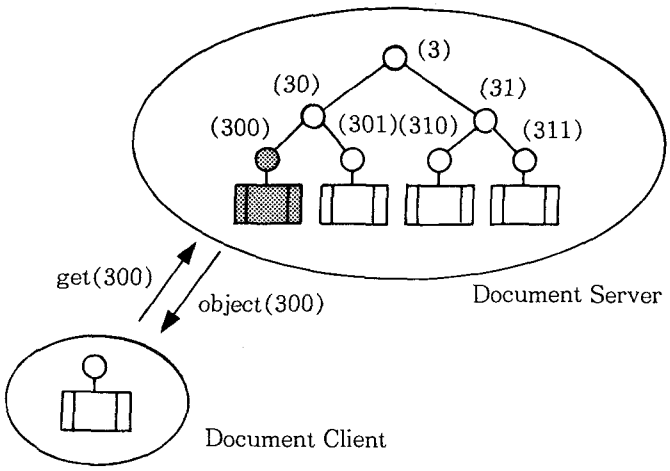
CDH(Cooperative Document Handling)와 같이 ODA문서내 정보와의 상호작용이 많이 필요한 응용에서는 내용부내의 매개의 데이타를 다루어야할 필요성이 높으며, 따라서 내용부내에서 처리위치 지정을 위한 메카니즘이 요구된다.

1. 객체 식별 메카니즘에 따른 문서처리

ODA문서를 구성하는 객체들은 객체식별자를 사용하여 각각 식별될 수 있다. 이러한 객체 식별 메카니즘은 (그림 1)에서와 같이 원격에 존재하거나 분산된 문서를 부분적으로 처리하기 위한 과정에서 중요한 역할을 담당한다.

(그림 1)에서 client는 server에게 (300)으로 식별되는 패러그래프를 요청하여 이를 수신한다.

작성자 : 함진호 선임연구원, 현동환 연구원(ETRI 프로토콜구조연구실)



(그림 1) 객체 정보 요구의 예

2. 내용부내에서의 위치지정 메카니즘의 필요성

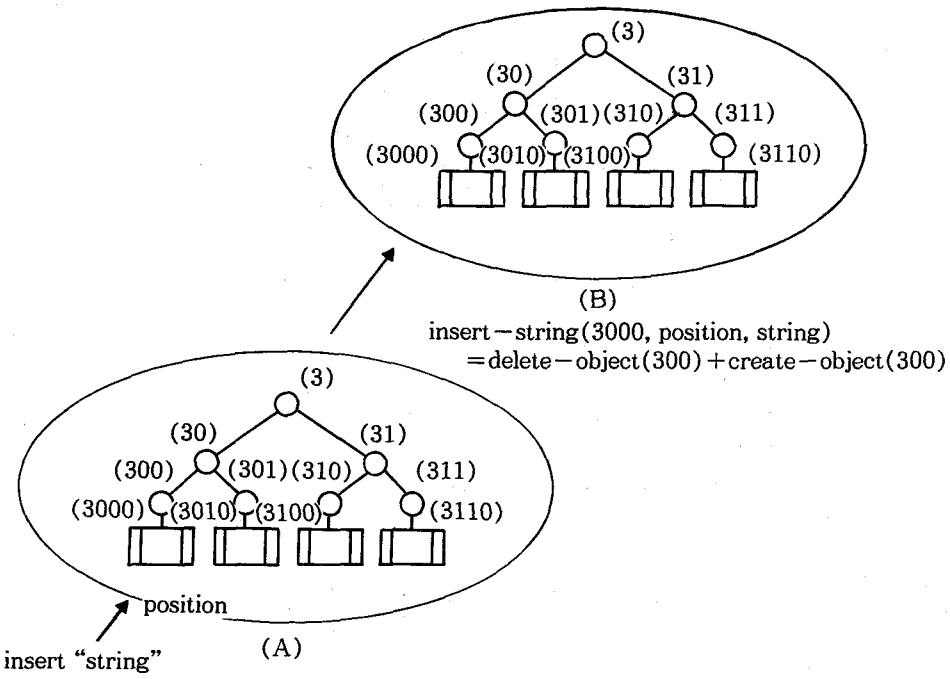
현대 ODA에 근거한 다양한 응용들이 등장하고 있으며, 그중의 하나가 CDH(Cooperative Document Handling)와 같은 응용이다. CDH에서의 문서처리는 객체단위가 아니라 기본 객체에 연결된 내용부내의 정보일 경우가 많다. 이러한 예로는 텍스트 편집에서의 문자입력, 공유된 이미지의 부분적인 수정등이 있을 수 있다. 이와같은 조작은 객체 단위의 식별 메카니즘만을 이용하여서는 원활히 수행될 수 없으며 많은 오버헤드를 수반한다. (그림 2)에서 문자입력의 예를 보였다.

현재 객체 식별 메카니즘만이 존재하고, 내용부내에서의 위치지정을 위한 메카니즘이 존재하지 않기 때문에 패러그래프 (300)에 어떤 문자열이 삽입되었다는 사실을 알려 A와 B의 문서를 동기화(synchronize)시키기 위하여는 B시스템에 해당 패러그래프를 제거하라는 메세지를 전달하고 A시스템에서 스트링을 삽입하여 만들어진 패러그래프 (300)을 B시스템의 문서의 필요한 위치에 다시 생성하라는 메시지를 전달하여야 한다. 이러한 문제는 문자 정보보다는 이미지정보와 같이 정보량이 많은 경우에 보다 심각해질 수 있다.

이와 같은 경우, 만일 내용부내의 특정위치를 지정할 수 있는 메카니즘이 존재한다면 단순히 내용부 (3000)의 특정위치에 추가된 스트링만을 삽입하라는 메세지를 전달함으로써 위에서 언급한 과정과 동등한 처리가 수행될 수 있다.

3. 결론

ODA문서조작에 있어서 보다 세련된 조직을 수행하기 위하여는 내용부 내에서 특정위치를 지정하기 위한 메카니즘이 필요하다.



(그림 2) character 삽입의 예

내용부 내의 위치지정은 내용 정보유형에 의존적이다. 따라서 내용정보의 유형별로 내용부내의 위치지정 메카니즘이 결정되어야 한다.

UIT- Secteur de la normalisation des telecommunications
ITU - Telecommunication Standardization Sector
UIT - Sector de Normalizacion de las Telecomunicaciones

Commission d'etudes)
Study Group) 8
Comision de Estudio)

Contribution tardive)
Delayed Contribution) D
Contribucion tardia)

Geneve, 16-25 November 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 3,15/8

SOURCE: KOREA (REPUBLIC OF)

TITLE: The need of positioning mechanism within content portion of ODA document for precise manipulation

---

Contact Point: Mr. Jin-Ho Hahm
Mr. Don-Whan Hyun
ETRI
P.O.Box 8, Daeduk Science Town
DAEJEON, 305-606, KOREA
Tel: +82 42 860 6115
Fax: +82 42 861 5404
Email: jhhahm@spring.etri.re.kr

ABSTRACT

An ODA document is composed of the hierarchical structure of the object. Each object is identified by object identifier within a document. Actual information of a document is contained in a content portion which is attached to a basic object. But currently, ODA base standard does not support the mechanism to specify the particular position within the content portion.

Many applications such as CDH(Cooperative Document Handling) require interactive processing of the information contained in a content portion. So, positioning mechanism enabling the specification of particular position within the content portion is required.

1. Object identification mechanism

Object in an ODA document is identified by the object identifier. This mechanism plays on an important role especially in partial manipulation of remote or distributed document. Fiqure 1, which illustrates the object identification mechanism, shows the fact that a client requests the paragraph identified by the object identifier (3 0 0).

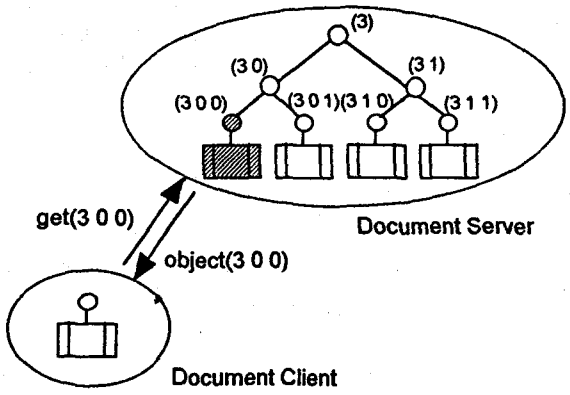


Figure 1 Example of the object fetch

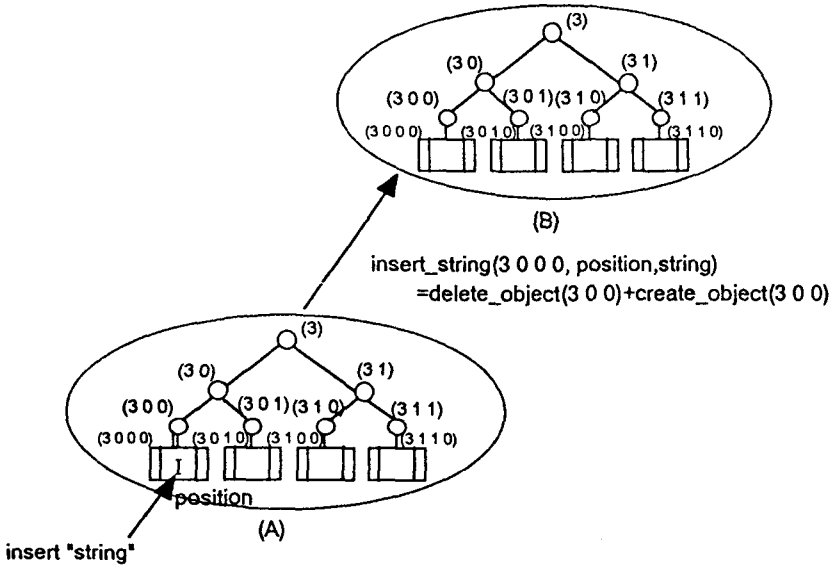
2. A need for positioning mechanism within a content portion

Recently, various applications, based on the ODA are developing. CDH(Cooperative Document Handling) is one of such applications. In Such a CDH application document handling had better be performed, not by the unit of the object, but by the unit of information whithin the content portion which is attached to the basic object. The insertion of the character string and the partial modification of the image in a shared document are related examples. Precise manipulations cannot be achieved by identifying the object unit only.

Figure 2 illustrates the example of the character string insertion. If we have only object identification mechanism and don't have any positioning mechanism within content portion, following manipulation can be a burden. With document A and B, when certain character string is inserted into the paragraph (3 0 0) of A, then in order for B

to be synchronized with A, we should send a message to delete the related paragraph and create modified paragraph (3 0 0) to B. The problem can be even heavier when dealing with bigger data like image.

In this case, if we have yhe mechanism to designate yhe particular position within a content portion, the procedure mentioned above will be replaced by the request to insert a string to a designated position within the content portion.



Fiqure 2 Example of the character insertion

3. Conclusion

To perform the precise manipulation of the ODA document, positioning mechanism within content portion is necessary.

Positioning mechanism is dependent on the information type of content. So, different positioning mechanism should be considered according to the type of content information.

UIT—Secteur de la normalisation des telecommunications
ITU—Telecommunication Standardization Sector
UIT—Sector de Normalization de las Telecommunications

Commission d'etudes)
Study Group) 8
Commision de Estudio)

Contribution tardive)
Delayed Contribution) D.
Contribution tardia)

Geneve, 16—25 November 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 15/8

SOURCE : KOREA(REPUBLIC OF)

TITLE : CDH(Cooperative Document Handling)서비스의 실현을 위한 집중방식 모델과 복제방식 모델의 비교

ABSTRACT

현재 CDH와 같은 공동문서작업은 일반적으로 집중방식 모델(centralized model)이나 복제방식 모델(replicated Model)에 따라 실현가능한 것으로 여러 연구에서 나타났다. 이러한 모델은 문서 공동작업의 형태, 편집하려는 문서의 양, 통신모델, 네트워크 환경 등에 따라 장단점을 가진다. 본 문서의 각 모델의 특징과 장단점들을 비교 검토함으로써 CDH와 DTAM의 모델 수립과 표준화 작업에서 고려하도록 하기 위한 것이다.

1. 집중방식 모델과 복제방식 모델

○ 집중방식모델 : 문서의 화일이 중앙의 한 시스템에만 존재하며 (그림 1), 공동작업을 수행하는 각 시스템에서는 화면이동이나 문서편집과 같은 작업시에 중앙에 보관된 문서의 해당부분을 그 때 그때 이용하는 방식이다. 그림 1에서 A와 B는 문서의 첫번째 패러그래프를 센터로부터 전송받아 갖고 있으며, C는 문서의 두번째 장(chapter)를 전송받아 가지고 있다. A, B, C 각 단말기에서는 전송받아 갖고 있는 해당부분을 화면에 디스플레이한다. 이경우 A가 패러그래프의 내용을 편집하게 되면, 그 내용을 센터에 알려 수정토록 한다. 이러한 과정은 흔히 클라이언트—서버(client—server)모델로 알려져 있다.

그러나 A, B, C의 문서공동작업이 원활히 수행되기 위해서는 특정부분이 수정되었다는 사실과

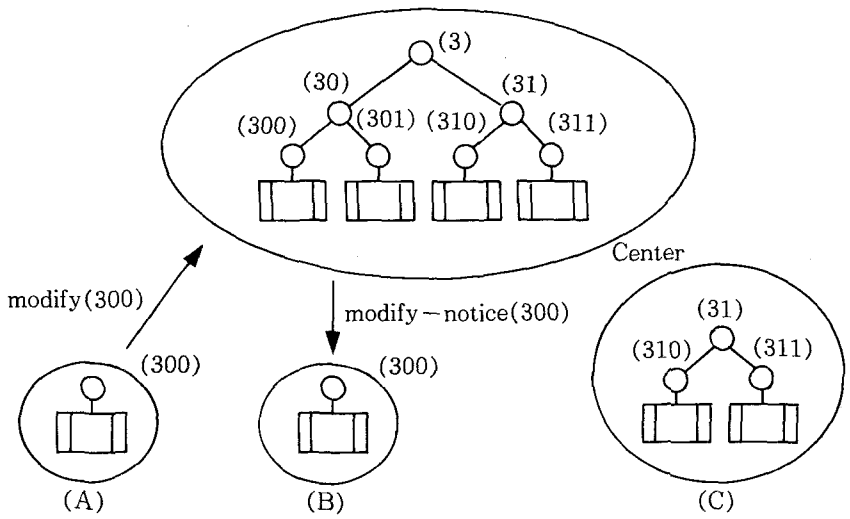
작성자 : 함진호 선임연구원, 현동환 연구원(ETRI 프로토콜 구조 연구실)

수정된 내용을 보고받은 센터는 이에대한 정보가 필요한 다른 단말(그림의 경우에는 B이다. A와 B는 현재 같은 부분을 보고있다.)에 이 사실을 즉시 통보할 필요가 있다. 만일 통보치 않는다면 A와 B는 문서의 같은 부분에 대하여 서로 다른 내용을 보고 있게 된다. 따라서 동기화된 원활한 문서의 공동편집을 위해서는 센터로부터 단말기로의 이와같은 내용의 통보가 필수적이다.

그러나 일반적으로 클라이언트-서버모델에서는 서버가 클라이언트에게 어떤 작업을 직접 요구할 수 없으므로 B는 바뀐 내용을 알기 위하여 센터에 이 사실을 요구하여 확인하여야 하므로, 이러한 과정을 통하여서는 실시간의 문서 공동작업이 수행될 수 없다. 따라서 (그림 1)에서의 처리 과정은 클라이언트-서버모델이라고 할 수 없으며 집중방식 모델이라고 불리워져야 한다.

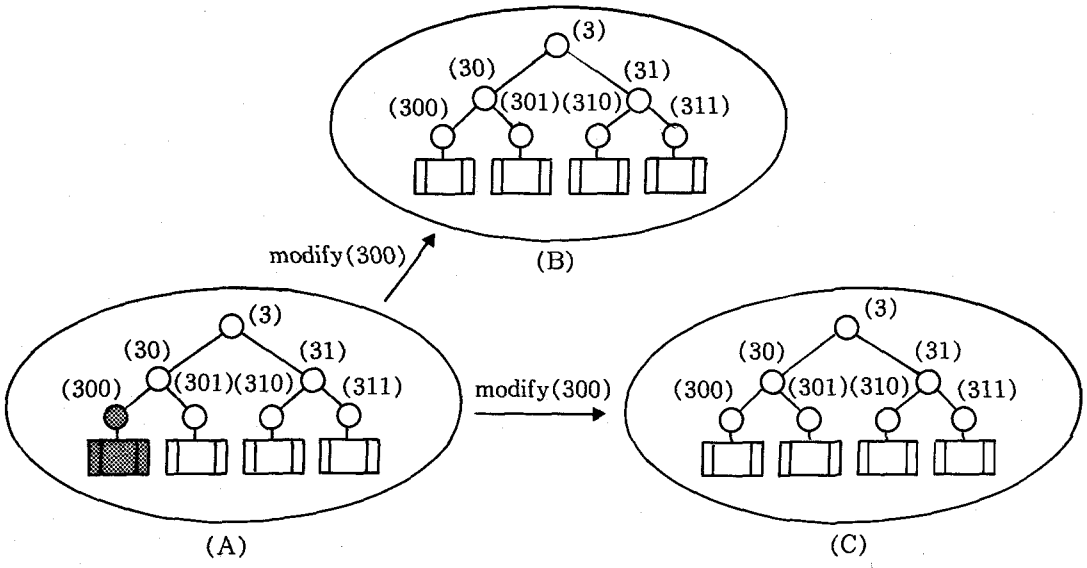
○ 복제방식 모델 : 공동으로 작업되는 문서의 화일이 모든 시스템에 복제되어 존재하는 방식으로 (그림 2), 공동작업을 수행하는 모든 시스템은 화면의 브라우징(*browsing*)을 위하여는 복제된 자신의 화일을 이용하고, 문서 편집의 경우에는 복제된 자신의 문서의 내용을 수정하고 이 사실을 즉시 다른 시스템에 통보하여 복제토록하는 방식이다.

(그림 2)에서 A시스템이 첫번째 패러그래프를 수정하게 된면, 이 사실을 같은 내용의 문서를



(그림 1) 집중방식 모델

복제하여 가지고 있는 B와 C시스템에 통보하여 수정하도록 요구한다. 이러한 과정을 통하여 A, B, C시스템은 동기화된 복제문서를 항상 소유하게 된다. 이러한 모델에서는, 공동 편집 작업 이전에 문서전체의 내용이 각 시스템으로 복제 되어야 한다.



(그림 2) 복제방식 모델

2. 각 모델의 장단점 비교 및 문제점 분석

위에서 설명한 집중방식 모델과 복제방식 모델은 문서공동편집에 있어서 한마디로 어떤 모델이 적합하다고는 할 수 없다. 이러한 모델은 문서공동작업의 형태, 문서의 양, 편집의 양, 통신모델, 네트워크 환경에 따라 장단점을 가진다. 여기에서는 이와 같은 문제점들을 비교 검토함으로써, CDH와 DTAM의 모델 수립과 앞으로의 표준화 작업에 반영하도록 한다.

2.1 집중방식 모델

장점

- 문서의 원본이 중앙 한곳에 존재하므로 문서의 일관성(consistency)유지에 유리하다.
- 문서의 양이 방대한 경우에도(멀티미디어 문서의 경우에 하나의 문서가 수~수십 메가바이트가 넘을 수도 있다) 문서의 전체내용을 각 시스템에 복제할 필요가 없다.
- 위와 같은 이유로하여, 문서 복제의 준비시간 없이 즉시 공동편집에 들어갈 수 있다.
- 문서 공동작업 중간에 참여가 간단하다.

단점

- 문서의 내용을 브라우징하는 과정에서도 끊임없이 센터에서 필요한 내용을 가져와야 하므로 네트워크 트래픽이 높다.
- 문서의 브라우징에서 네트워크 트래픽이 요구됨에 따라 응답시간이 지연되어 사용자에게 지루한 감을 줄 수 있다. 이를 극복하기 위하여는 높은 전송 대역폭이 제공되어야 한다.

- 센터측의 부담이 단말기에 비하여 많으며, 센터로서의 역할을 공동작업에 참여하는 단말기중의 하나가 담당하게 될 경우, 특정 단말기의 부담이 과중하게 된다.
- 공동작업이 완료된 문서를 각자 소유하기를 원할 경우, 공동편집 완료후 센터의 문서화일을 각 단말기로 복제하는 과정이 필요하다.

2.2 복제방식 모델

장점

- 공동편집에 참여하는 시스템들의 부하가 균등하게 배분된다.
- 문서의 전체내용을 이미 각 단말기에서 복제하여 가지고 있으므로, 문서의 브라우징에 트래픽이 발생하지 않는다.
- 문서의 브라우징에 트래픽이 발생하지 않으므로, 응답시간이 빠르다.

단점

- 복제된 문서의 일관성을 유지하기 위해서는 통신프로토콜이 복잡해진다.
- 문서편집에 앞서, 문서를 모든 시스템으로 복제하여야 하므로 준비시간이 소요된다. 이러한 이유로 방대한 양의 문서에 대하여 적은 편집만이 수행되고, 공동편집에 참여한 사람들이 편집결과를 당장 소유하기를 원하지 않을 경우 비경제적이다.
- 공동작업과정에서의 중도 참여를 위한 프로토콜이 복잡하다.

3. 결론

앞에서 CDH와 같은 공동작업을 수행하는 집중방식 모델과 복제방식 모델의 특징에 대하여 살펴보았으며, 각 모델의 장단점에 대하여 검토하였다. 검토결과 어떤 특정모델이 CDH와 같은 응용에 더 적합한지는 계속적인 검토가 필요하다. CDH 어플리케이션이 두가지 모델을 갖는 것이 바람직하지 않으므로 위의 장단점을 검토하여 하나의 모델로 통합하는 것이 바람직하다고 생각된다.

UIT- Secteur de la normalisation des telecommunications
ITU - Telecommunication Standardization Sector
UIT - Sector de Normalizacion de las Telecomunicaciones

Commission d'etudes)
Study Group) 8
Comision de Estudio)

Contribution tardive)
Delayed Contribution) D
Contribucion tardia)

Geneve, 16-25 November 1993

Texte disponible seulement en)
Text available only in) E
Texto disponible solamente en)

Question : 15/8

SOURCE: KOREA (REPUBLIC OF)

TITLE: The Comparison between Centralized Model and Replicated Model for Realization of CDH(Cooperative Document Handling) Service

---

Contact Point: Mr. Jin-Ho Hahm
Mr. Don-Whan Hyun
ETRI
P.O.Box 8, Daeduk Science Town
DAEJEON, 305-606, KOREA
Tel: +82 42 860 6118
Fax: +82 42 861 5404
Email: jhhahm@spring.etri.re.kr

ABSTRACT

Many researches show that cooperative editing works such as the CDH can be realized in two models, i.e., the centralized model and the replicated model. The two models have reversed advantages and disadvantages concerning the points like the types of cooperative document processing, document size, communication model, and network environment.

In this document, we compare and analyze two models on the above points. This is to be considered as a meaningful input for constructing ideal models of the CDH and the DTAM(Document Transfer and Manipulation), and also for the standardization works.

1. Centralized Model and Replicated Model

o Centralized Model

Document file is stored in a central system as in the Figure 1. All the systems of the cooperative work use the same document stored in the central system. Whenever they edit or browse, they use only that portion of the document stored in the central system.

In the Figure 1, system A and B requested and received the first paragraph of the document stored in the central system, while C received second chapter. All the systems A, B, and C can display their working portions on their monitors. When system A edit its paragraph, that fact is delivered to the central system. This processing model is often regarded as a client-server model. But in this model, the server can not request services from the client. However, in the Figure 1, in order for A, B, and C to work together fluently, any editing operation known to the central system should be automatically delivered to other related systems (system B in the Figure 1). Without this process, A and B would be dealing with different contents. In the case of the client-server model, to keep consistency of the displayed portions between A and B, the system B in the Figure 1 should request to the central system to check that there is any modification.

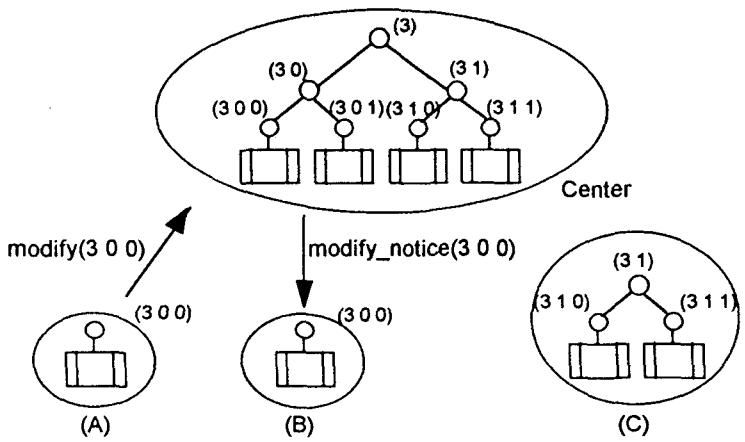


Figure 1 Centralized Model

With this kind of checking process, synchronized real-time cooperative document processing is hard to be achieved. So, the client-server model is not enough and

centralized model performing automatic delivering of every update operations to other systems is required.

o Replicated Model

Document file is copied and stored in all the systems as in the Figure 2. Editing and browsing within the document is performed on their own replicated document file and this information is transferred to other systems.

In the Figure 2, when system A modify its first paragraph, system A transfer this fact(modifying the first paragraph) to system B and C so that these systems can update their own documents together. With this process, system A, B, and C can maintain synchronized replicated document all the time. In this model, whole document is copied to all the other systems before a cooperative work begins.

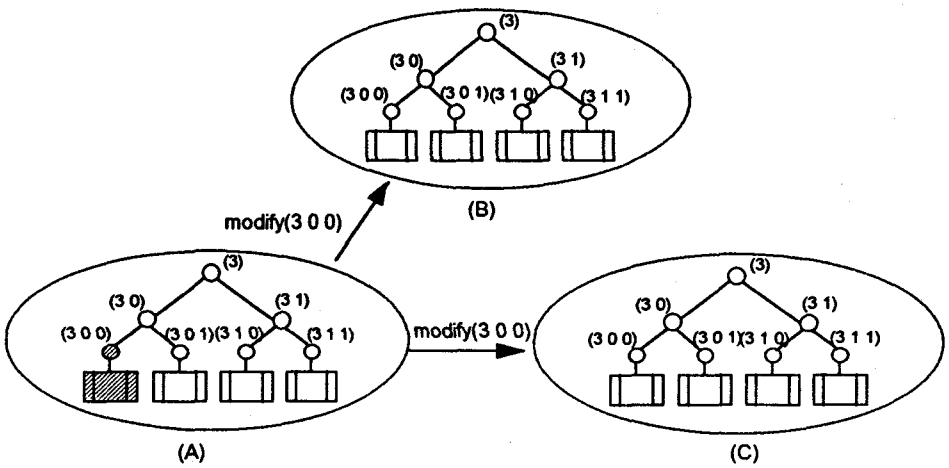


Figure 2 Replicated Model

2. Comparison Analysis between Two Models

One can not say that which one is more suitable to the general cooperative document processing. Two models have reversed advantages and disadvantages concerning the types of document handling, document size, editing size, communication model, and network environment.

Through the comparison analysis on these concerning points, establishment of the model of the CDH and the DTAM can be carefully considered.

2.1. Centralized Model

Advantages

- Maintenance of Document Consistency is Simple. (Master document is stored only one central system)
- Less Resource Are Required (Replication of the document is not necessary, so memories of other systems are saved)
- Document Replication Time is not necessary(Without replication)
- Joining in the Middle of an Editing Work is Simple (Document Replication is not necessary)

Disadvantages

- High Network Traffic (Even on browsing the document, each terminal should fetch necessary contents from the central system)
- High Bandwidth is Required (Browsing takes a network traffic which results in delayed responses and makes user board)
- Heavy System Load on Central System (When the role of central system is designated on the one of the terminals, that one takes much system overhead)
- Additional Document Fetching Process is required after the Completion of Editing (Terminals have only partial documents in the course of an editing)

2.2. Replicated Model

Advantages

- Even Distribution of the System Load among All the Systems
- No Network Traffic on Browsing (Whole document is replicated on every system, so each can browse their own document file)
- Shorter Response Time on Browsing

Disadvantages

- Complex Communication Protocol (Complex protocol is required to keep the consistency of all the replicated documents)

- Longer Document Preparation Time is Required (Document replication takes time. In the case of little editing on a large document it is uneconomical.

3. Conclusion

The centralized model and the replicated model for the cooperative editing work like CDH are discussed and advantages and disadvantages of each model is also carefully analyzed. Decision on the model for the cooperative work can not be clearly made and still need futher consideration. One possible way is containing both models together and selectively applying a better one on each case. But, it would be better to establish a single combined model. A unified model containing advantages of both models seems to be a proper answer. This unified model will have close relationship with the design of the CDH and DTAM structure.