数理解析研究所講究録
第809巻 1992年 184-196

A State Space Representation of Business Transaction System

Ryo SATO
Institute of Socio-Economic Planning,
University of Tsukuba,
Tsukuba city, Ibaraki 305, Japan

abstract

Functions of enterprises are usually considered to consist of three levels: administration, management, and operation levels. In this paper a causal model for systems of operation level is proposed. Intuitively speaking an operation system is a discrete event system. That is, for example, orders or enquiries from customers come in, purchase orders go out, goods are produced or bought into warehouse, and they are sent with invoices. Each of those activities can happen randomly and in parallel with respect to time and place, and changes in discrete way. The model is a state space representation whose state space consists of files and internally set events of some activities.

There are identifiable amount of information systems methodologies. Every methodology does not seem to have concrete general models of operation system to which information systems are supposed to be a part of and/or support. That situation is strange because no one can design highly suitable systems if he does not have a model of his target system. The model in this paper will play such role and add some meaning of documents and activities in those methodologies.

1. Introduction

Operation system in an enterprise can be taken as a discrete event system. For example, it processes orders, production, purchase, bookkeeping, e.t.c. in parallel way and at random. The questions to be answered in this paper are:

i) What decides the dynamics of operation system?

ii) What kind of roles does a file system play in a business system?

iii) What kind of systems do many information systems methodologies produce?

The model will provide conceptual basis to business analysis and requirements identification in information systems methodologies, although we do not develop those kind of implication to methodologies in this paper.

Model building is two fold. The model of dynamics is a state space representation of operation system as a discrete event system whose state space consists of files and internally set events of some activities. The activities change the value of files. In modelling a real world situation from static information aspect, data is used and modelled by files with attributes and two types of integrity conditions. This modelling is called data modelling.

2. Basic Concepts

In this section systems, basic concepts and some notation are defined according to Mesarovic and Takahara[2].

Definition 1. *system*

A system is a relation of an input set and output set. If those two sets are sets of time functions defined on the same time set, then the system is called a time system.

The value set of inputs of a time system is called input alphabet and that of output called output alphabet. Usually one of the set of non-negative real numbers R^+ or that of non-negative integers Z^+ is taken as a time set. Let x be a function from a time set T to a input alphabet. For any $t, t' \varepsilon T, t \leq t'$, the restriction of domain of x to $[t, t')$ is written as $x_{tt'}$. That is, $x_{tt'}(\tau) = x(\tau)$ for any $\tau, t \leq \tau < t'$. Similarly x_t is defined as $x_t(\tau) = x(\tau)$ for any $\tau, t \leq \tau$. Let x and x' be arbitrary functions from T to the same alphabet A. For any $t \varepsilon T$, we can define another element $x''\colon T \to A$ by

$$x''(t'') = \begin{cases} x(t''), \text{ if } t'' < t \\ x'(t''), \text{ if } t'' \geq t \end{cases}$$

x" is called the concatenation of x_{0t} and x'_t and denoted by $x'' = x_{0t} \cdot x'_t$. We define input space X of S by $X = \{x \mid (\exists y)((x, y) \varepsilon S)\}$. The output space Y of S is defined similarly. The power set of a set A is denoted by P(A), that is, $P(A) = \{A' \mid A \supseteq A'\}$.

State space is a key concept by which we can grasp the behavior of the dynamic system.

Definition 2. *state space representation*

Let $S \subset X \times Y$ be a time system with output alphabet B. A pair $<\underline{\phi}, \mu>$ where

$$\underline{\phi} = \{\phi_{tt'} \mid \phi_{tt'} : C \times X_{tt'} \to C \text{ and } t, t' \varepsilon T, t \leq t'\}$$

and $\mu : C \to B$

is a state space representation of S if and only if the following conditions are satisfied:

(i) the functions $\underline{\phi}$ satisfies the following

(α) $\phi_{tt''}(c, x_{tt''}) = \phi_{t't''}(\phi_{tt'}(c, x_{tt'}), x_{t't''})$, where $t \leq t'' \leq t'$ and $x_{tt''} = x_{tt'} \bullet x_{t't''}$

(β) $\phi_{tt}(c, x_{tt}) = c$

(ii) $(x,y) \varepsilon S$ if and only if there exists some $c \varepsilon C$ such that for any $t \varepsilon T$

$$y(t) = \mu(\phi_{0t}(c, x_{0t})).$$

C is called the state space of $<\underline{\phi}, \mu>$.

Especially $\underline{\phi}$ is called a transition family if it satisfies (α) and (β) of the above definition.

State space representation is the wide-spread framework to recognize dynamics of a time system in causal way. Mesarovic and Takahara[2] shows that a time system is causal if and only if it has a state space representation.

A dscrete event system is a special time system defined as follows:

Definition 3. *discrete event system [4]*

If a time system $S \subset X \times Y$ satisfies the following four conditions then is called a discrete event system:

1) $T=[0, T_{end})$, $T_{end} \varepsilon R^+$;

2) There is a set A' and the input alphabet A of input space X is the power set of A'. That is, $A = P(A')$;

3) For any $(x, y) \varepsilon S$ the following two conditions holds:

3-1) $event(x) = \{t \mid x(t) \neq 0, 0 \text{ is the empty set}\}$ is finite;

3-2) $F(y) = \{[t, t') \mid y(t) = y(t'') \text{ for any } t'', t \leq t'' < t', \text{ and } y(t) \neq y(t')\}$ is finite and $\cup F(y) = T$.

We will see a transaction system as a discrete event system.

3. Objects for data modelling

3.1 Sets for data modelling

We will use DAE (Domain - Attribute - Entity) datamodel for description of data structure. DAE is a simplification of TH datamodel[1]. The most famous datamodel seems to be entity-relation(ER) model. The reason why we adopt a DAE datamodel instead of ER is that the natural correspondence between states and files, which is defined below, and the concept of relation needs not to be indispensable. Intuitively state at time is the historical and internal information of the time system that decides future response of the system. The file system records accumulation of past transactions and seems to be a part of the state.

Definition 4. *definition of sets*

ES :the set of entities called entity set.

ETS :set of entity types called entity type set. ETS is finite.

DS :set of domains called domain set. DS is finite and a subset of ETS.

KS :set of key entity types called key entity type set. KS is finite and a subset of DS. A key entity type is also called control object.

FTS :set of file types called file type set. FTS is finite and a subset of ETS.

AS : set of attributes called attribute set.

LES : set of list entities called list entity set.

ETS = DS$\cup$FTS(direct sum)

S: DS $\rightarrow$ P(ES); set valued function

ES = ETS$\cup$LES$\cup$AS$\cup$S(DS)

LES = $\cup\{P(S(dom(a_{i_1}))\times\ldots\times S(dom(a_{i_n}))) \mid A(FT(x)) = \{a_{i_1}, \ldots, a_{i_n}\}$ for some $x \in KS\}$

3.2 Functions for data modelling

Definition 5. *definitions of functions*

FT : KS $\rightarrow$ FTS; file type function

FT(x) is called the file type of x, which is of key type.

A : FTS $\rightarrow$ P(AS); attribute function. When $A(FT(x))=\{a_1, a_2, \ldots, a_n\}$ then each a_i in A(FT(x)) is called the attribute of FT(x).

dom: AS $\rightarrow$ DS; domain function. dom(a) is called the domain of a.

key: FTS $\rightarrow$ AS, such that

(1) If y = FT(x) then dom(key(y)) = x, and

(2) dom(key(FT(x))) = x is always holds.

File = {f: FTS $\rightarrow$ LES | f is a file-content function}, where f is said to be a file-content function if it satisfies for any x in KS that f(FT(x)) is a subset of $S(dom(a_{i_1}))\times\ldots\times S(dom(a_{i_n}))$ and A(FT(x)) = $\{a_{i_1}, \ldots, a_{i_n}\}$. f(FT(x)) virtually represents the set of tuples in FT(x) at a time.

For an arbitrary file-content function f define V_f: f(FTS)$\times$AS $\rightarrow$ S(DS) as follows;

Let $A(FT(x))=\{a_1, a_2, \ldots, a_n\}$.

For any $y \in S(FT(x))$ and $a_i \in AS$, $V_f(y, a_i) \in S(dom(a_i))$ holds.

The function V_f is often written as V when the file-content function f is obvious in the context.

Fig. 1 shows some relation between the above defined sets and functions.

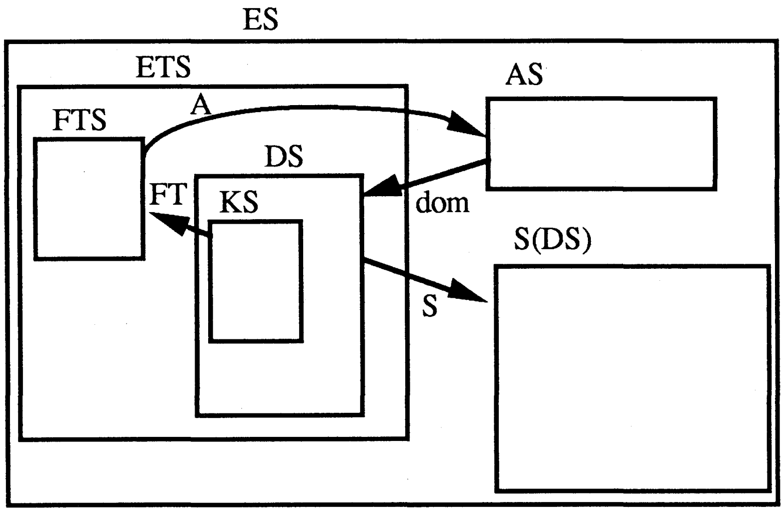

Fig. 1 Domain - Attribute - Entity detamodel

3.3 Two types of integrity conditions

Definition 6. *Referential key (referential integrity)*

Let x_1 and x_2 be arbitrary elements of KS. An attribute $a_i \in A(FT(x_2))$ is called *a referential key of $FT(x_2)$ to $FT(x_1)$* of *$FT(x_2)$ refers $FT(x_1)$* iff

1) $dom(key(FT(x_1))) = dom(a_i) = x_2$ hold, and

2) For any $f \in$ File

$f(FT(x_2)).key(FT(x_1)) \supseteq f(FT(x_2)).ai$ holds.

In the above case we will write as $FT(x_2) \leftarrow FT(x_1)$.

Integrity shown by referential keys is called referential integrity.

REFER relation

A binary relation *REFER* on AS is defined as;

$(a, b) \in$ REFER iff there exist x_1 and x_2 in KS such that

1) $a \in A(FT(x_2))$ and $b = key(FT(x_1))$, and

2) a is a referential key of $FT(x_2)$ to $FT(x_1)$.

Definition 7. *Subtype and Supertype*

Let x_1 and x_2 be arbitrary elements of KS. $FT(x_1)$ is said to be a *subtype* of $FT(x_2)$ or $FT(x_2)$ is *supertype* of $FT(x_1)$ iff the following three conditions hold;

1) For any $f \in$ File, $V(f(FT(x_2)), key(FT(x_2))) \supseteq V(f(FT(x_1)), key(FT(x_1)))$; That is, for any tuple y in file $f(FT(x_1))$ there exists a tuple $z \in f(FT(x_2))$.

2) $A(FT(x_1))-key(FT(x_1)) \supseteq A(FT(x_2))-key(FT(x_2))$; That is, $FT(x_1)$ has more attributes than $FT(x_2)$. In other words $FT(x_1)$ inherits the attributes of $FT(x_2)$.

3) For arbitrary f $\in$ File, $y_1 \in f(FT(x_1))$ and $y_2 \in f(FT(x_2))$ if $V(y_1, key(FT(x_1))) = V(y_2, key(FT(x_2)))$ then $V(y_1, a) = V(y_2, a)$ for any $a \in A(FT(x_2))$-$key(FT(x_2))$; That is, the corresponding values of inherited attributes are the same if that of key attributes so.

If $FT(x_1)$ is a subtype of $FT(x_2)$ then we denote as

$$FT(x_1) \leftarrow\!\bullet\!- FT(x_2)$$

or *$FT(x_1)$ inherits $FT(x_2)$.*

INHERITS relation

A binary relation *INHERITS* on FTS is defined as
(a, b) $\in$ INHERITS iff a inherits b.

REFER's and INHERITS' are used to build in a logical conditions that should hold in files in any implementation.

Let ¢FTS, ¢AS, ¢DS, ¢INHERITS and ¢REFER be special elements of FTS such that for any file-contents function f f(¢FTS) = FTS, f(¢AS) = AS, f(¢DS) = DS, f(¢INHERITS) = IHHERITS and f(¢REFER) = REFER hold. Then the integrity conditions that DAE datamodel must satisfy are shown as Fig. 2.

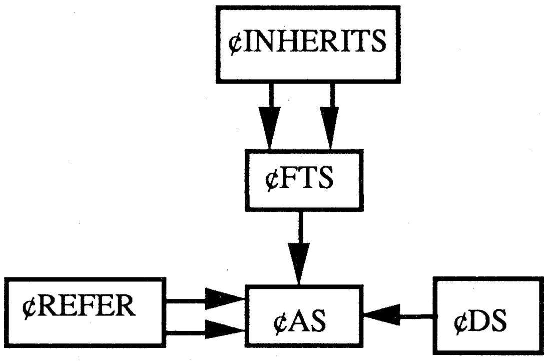

Fig. 2 integrity conditions in DAE

4. Business Transaction System

In the following a model of transaction system as a discrete event system with file system is formulated. And it will be proved to be a state space representation (discrete event dynamical system) of the discrete event system. The dynamic mechanism of the state space representation is based on the three phase approach[3] to discrete event simulation that was formulated in [4]. Detailed intention of notions will not be explanation in the following definition. Explanation of them can be seen in [4].

In the following /*...*/ is a comment.

Definition 8. *business transaction system* $<\phi, \Pi_F>$

8-1) *definition of auxiliary sets*

Clock = $[0, T_{end}]$, where $T_{end} \varepsilon R^+$ /* time set of three phase simulation system */

$\Delta \varepsilon$ Clock /* The smallest time-slice to proceed simulation */

Entity = $\{1, 2, ..., n\}$ /*Every entity has a unique name as a number. An entity acts on files*/

Entity = B_PEntity $\cup$ B_BEntity $\cup$ B_CEntity

, where B_PEntity, B_BEntity and B_CEntity are mutually disjoint.

AnEntityState = TimeCell$\times$NextActivity$\times$Avail

,where TimeCell = Clock

NextActivity = B_PActivity $\cup$ B_BActivity $\cup$ B_CActivity

Avail = {available, void} /*shows whether the time cell of the entity state is currently available or not*/

and B_PActivity, B_BActivity and B_CActivity are finite and mutually disjoint.

/* Each entity is associated to an activity. If the activity is an element of Activity and the value of Avail is "available" then the activity will occur at the time represented in TimeCell. If the activity is of CActivity possibility of occurrence of it is examined at each time any activity occurred and executed in C_Phase below. */

Activity = NextActivity $\cup$ CActivity

, where CActivity = $\{c_1, c_2, ..., c_w\}$ is finite and disjoint from NextActivity.

EntityStates: Entity $\rightarrow$ AnEntityState /* each entity has its state */

B_BEntityStates is the restriction of EntityStates to B_BEntity.

B_CEntityStates is the restriction of EntityStates to B_CEntity.

CEntityStates = B_BEntityStates $\cup$ B_CEntityStates

FileId = KS = $\{1, 2, ..., u\}$ /* each file in the system has its name as a number */

$f_{AfectFile}$: Activity $\rightarrow$ P(FileId) /* Each activity affects files specified by $f_{AfectFile}$. */

X = $\{x \mid x : \text{Clock} \rightarrow P(B_P\text{Activity}), \{t \mid x(t) \neq 0, 0 \text{ is the empty set}\} \text{ is finite}\}$ /* input */

BTL = $\{f \mid f : B_P\text{Activity} \rightarrow [0, T_{end}]\}$ /*Note that L(x)(t) is an element of BTL for any $t \varepsilon [0, T_{end}]$ */

f_{NextB_BAct}: B_PActivity $\rightarrow$ B_BActivity, one-to-one mapping.

/* Each B_B activity is bootstrapped by its corresponding B_P activity, specified by f_{NextB_BAct}.*/

f_{BEA}: B_BEntity $\rightarrow$ B_BActivity ; one-to-one mapping.

$f_{PEA}$: B_PEntity $\rightarrow$ B_PActivity ; one-to-one mapping.

/* Occurrence time of B_P and B_B activities are set in its corresponding a B_P and B_B entities, respectively. Functions $f_{PEA}$ and $f_{BEA}$ specifies the correspondence.*/

f_{EE}: B_PEntity $\rightarrow$ B_BEntity, $f_{EE} = f_{BEA}^{-1} \bullet f_{NextB_BAct} \bullet f_{PEA}$.

f_{B_BEA} : B_BEntity $\rightarrow$ B_PActivity, $f_{B_BEA} = f_{NextB_BAct}^{-1} \bullet f_{BEA}$.

f_{NextB_CAct}: CActivity $\rightarrow$ B_CActivity : one-to-one mapping.

/* Each B_C activity is bootstrapped by its corresponding C activity, specified by f_{NextB_CAct}. */

/* If z is an element of a Cartesian product and has A-coordinate then the A-coordinate of z is written as "z.A". For example, if z = (a, b, c) ε A$\times$B$\times$C then z.A = a and z.B = b. Changes with time in EntityStates and File can be divided into three phase, each of which is named A_Phase, B_Phase, and C_Phase. */

8-2) *Transition in A_Phase*

f_{Scan}: EntityStates $\rightarrow$ Clock

f_{Scan}(e) = min {k | k = e(i).TimeCell and

e(i).Avail = available for some entity i ε Entity}

f_{Dues}: EntityStates $\rightarrow$ P(Entity), where P(Entity) is the set of subsets of Entity.

f_{Dues}(e) =

{i | f_{Scan}(e) = e(i).TimeCell, and e(i).Avail = available}

8-3) *Transition in B_Phase*

We define f_{B_Ent} as follows.

Let e ε EntityStates be arbitrary and d = f_{Dues}(e).

f_{B_Ent} : EntityStates $\rightarrow$ EntityStates

f_{B_Ent}(e) = e' such that for each i ε Entity

```
    case 1: if i ε d ∩ (B_B Entity ∪ B_C Entity) then
        e'(i).NextActivity = e(i).NextActivity /*unchanged*/
        e'(i).TimeCell = e(i).TimeCell /* unchanged*/
        e'(i).Avail = void.
    case 2: if i ε d ∩ B_P Entity then
        e'(f_EE(i)).NextActivity = f_NextB_BAct(e(i).NextActivity)
        e'(f_EE(i)).TimeCell =f_NextTime(c)(f_NextB_BAct(e(i).NextActivity))
        e'(f_EE(i)).Avail = available.
    case 3: when i is not in d:
```

e'(i) = e(i) /* unchanged*/

, where $f_{NextTime}$: Clock×(B_BActivity ∪ B_CActivity) → Clock and $f_{NextTime}(c)(f_{NextB_BAct}(e(i).NextActivity)) \geq \Delta + c$ holds.

f_{B_File}: EntityStates×File → File /* f_{B_File} calculates File when any of Activity occurs. */

$f_{B\_File}(e, f) = f_{nu}$, such that $f_{10} = f_{0u} = f$ and

$$f_{ij}(k) = \begin{cases} f_{FileVal}(a, f_{i-1,u}, k), \text{ if } i \varepsilon d, k \varepsilon f_{AfectQ}(a) \text{ and } k = j = 1; \\ f_{FileVal}(a, f_{i,j-1}, k), \text{ if } i \varepsilon d, k \varepsilon f_{AfectQ}(a), k = j \text{ and } 1<j\leq u; \\ f_{i-1,u}(k), \text{ if } i \notin d \text{ and } j = 1; \\ f_{i,j-1}(k), \text{ otherwise}; \end{cases}$$

for each i and j, $1 \leq i \leq n$, $1 \leq j \leq u$, where a = e(i).NextActivity and $f_{FileVal}$:Activity×File×KS → LES such that $f_{FileVal}(a', f', k') \in P(S(dom(a_{i_1}))\times\ldots\times S(dom(a_{i_n})))$ with $A(FT(k)) = \{a_{i_1}, \ldots, a_{i_n}\}$ for any a' ∈ A, f' ∈ File and k'∈ KS.

8-4) *transition in C_Phase*

$f_{C\_condition}$: File → P(CActivity)
f_{CB}: CActivity → B_CEntity : one-to-one mapping.

Let i ε Entity, f ε File and e ε EntityStates be arbitrary and $c = f_{Scan}(e)$.

f_{C_Ent}:File×EntityStates → EntityStates
$f_{C\_Ent}(f, e)$ = e' such that for any i ε Entity

```
    case 1: if $i \notin f_{CB}(f_{C\_condition}(f))$ or $f_{C\_condition}(f)$ is empty then
        e'(i) = e(i) /* unchanged */
    case 2: if $i \varepsilon f_{CB}(f_{C\_condition}(f))$ then
        e'(i).NextActivity = $f_{NextB_CAct}(f_{CB}^{-1}(i))$
        e'(i).TimeCell = $f_{NextTime}(c)(f_{NextB_CAct}(f_{CB}^{-1}(i)))$,
                    ,where $f_{NextTime}(c)(f_{NextB_CAct}(f_{CB}^{-1}(i))) \geq \Delta + c$ holds,
        e'(i).Avail = available
```

/* B_CActivity is another type of bootstrapped activity. In an inventory section for goods, purchased goods arrive randomly and they will stay in the warehouse until being shipped.

That kind of activity, for example "stay", is a C activity. When we think the length of holding goods varies to each customer's order, the time of shipping is determined when the stay activity starts. Thus the activity of "the end of staying" occurs at the set time. A type of activities such as "the end of staying" that is bootstrapped by a C activity is called B_CActivity. The occurrence time stored in a B_CEntity. */

f_{C_File}: File $\rightarrow$ File /* f_{C_File} calculates File when any of CActivity occurs. */

$f_{C\_File}(f) = f_{wu}$ such that $f_{0u} = f_{10} = f$ and

$$f_{ij}(k)=\begin{cases} f_{FileVal}(c_i, f_{i-1,u}, k), \text{ if } c_i \varepsilon f_{C\_condition}(f_{i-1,u}), k \varepsilon f_{AfectQ}(c_i) \text{ and } k = j = 1; \\ f_{FileVal}(c_i, f_{i,j-1}, k), \text{ if } c_i \varepsilon f_{C\_condition}(f_{i,j-1}), k \varepsilon f_{AfectQ}(c_i), k=j \text{ and } 1<k\leq u; \\ f_{i-1,u}(k), \text{ if } c_i \not\varepsilon f_{C\_condition}(f_{i-1,u}) \text{ and } j = 1; \\ f_{i,j-1}(k), \text{ otherwise}; \end{cases}$$

for any i, j, $1 \leq i \leq w$, $1 \leq j \leq u$.

8-5) *construction of* $<\phi, \Pi_F>$

f_B:File×EntityStates $\rightarrow$ File×EntityStates.

Let f ε File and e ε EntityStates be arbitrary.

$f_B(f, e) = (f_{B\_File}(e, f), f_{B\_Ent}(e))$. /* f_{Scan} and f_{Dues} are used in calculation of f_B. */

/* f_C is a total function which satisfies the following */

f_C : File×EntityStates $\rightarrow$ File×EntityStates

$$f_C(f, e) = \begin{cases} f_C(f_{C\_File}(f), f_{C\_Ent}(f, e)), \text{ if } f_{C\_condition}(f) \neq 0 \\ (f, e), \text{ if } f_{C\_condition}(f) = 0 \end{cases}$$

δ:File×EntityStates $\rightarrow$ File×EntityStates

$\delta = f_C \bullet f_B$

Let e^B ε B_PEntityStates and e^C ε CEntityStates be arbitrary. We will identify a pair (e^B, e^C) as an element of EntityStates in natural way.

δ_C: File×EntityStates $\rightarrow$ File×CEntityStates is defined by $\delta_C(f, e) = (f', e'^C)$ such that $\delta(f, e) = (f', e'^B, e'^C)$ for some e'^B ε B_PEntityStates.

f_{XEnt}:BTL $\rightarrow$ B_PEntityStates

$f_{XEnt}(\beta) = e'$ such that

e'(i).NextActivity = $f_{PEA}(i)$,

e'(i).TimeCell = $\beta(f_{PEA}(i))$,

e'(i).Avail = available

for any i ε B_PEntity.

$\phi_{tt'}$: File×CEntityStates×$L(X)_{tt'}$ → File×CEntityStates is defined by as follows:

for any t and t', t < t',

$$\phi_{tt'}(f, e^C, L(x)_{tt'}) = \begin{cases} (f_m, e^C{}_m), \text{ if } t_m \le t' \text{ and } t_{m+1} > t' \\ (f, e^C), \text{ otherwise,} \end{cases}$$

where $t_0 = t$, $(f_0, e^C{}_0) = (f, e^C)$, $t_k = f_{Scan}(f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C{}_{k-1})$ and $(f_k, e^C{}_k) = \delta_C(f_{k-1}, f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C{}_{k-1})$ for some positive integer n and each k, $1 \le k \le m+1$. Also define $\phi_{tt}(f, e^C, L(x)_{tt}) = (f, e^C)$ for any t.

The family of functions defined above $<\underline{\phi}, \Pi_F>$, where $\underline{\phi} = \{\phi_{tt'} \mid t, t' \in T, t \le t'\}$ and Π_F is the projection on File×CEntityStates along File, is called a business transaction system.

Well-definedness of $\phi_{tt'}$ must be shown. Since event(x) is finite we can assume that event(x) = $\{t_1, t_2, ..., t_p\}$, $t_1 < t_2 < ... < t_p$ for some integer p. At each t_i, $1 \le i \le p$, for some of CEntity, say j ε CEntity, CEntityStates(j).TimeCell is set by f_{B_Ent} and f_{C_Ent}, and for some of B_PEntity k B_PEntityStates(k).TimeCell by f_{XEnt}. Therefore if the number of Entity is r then at most r entities are set in EntityStates such that the state value of File×EntityStates changes at each of the set times. Thus the number of times that give is at most p∗r even for t = 0 and t' = T_{end}.

In this paper we have restricted our consideration in the case where f_C is a total function, that is, the expansion of f_C is eventually stops by $f_{C\_condition}$ value being empty.

For a three phase simulation system $<\underline{\phi}, \Pi_F>$ the *resultant system* that $<\underline{\phi}, \Pi_F>$ defines is denoted by Res($<\underline{\phi}, \Pi_F>$), that is, Res($<\underline{\phi}, \Pi_F>$) $=\{(L(x),y) \mid (\exists((f, e^C))(\forall t)\ (y(t) = \Pi_F \bullet \phi_{0t}(f, e^C, L(x)_{0t}))\}$.

Definition 9. *L (time-list representation) [4]*

Let x be an input of a discrete event system (A', B, T, X, Y, S). Notice x(t) ε P(A') holds for any t ε T.

Define L: X → {T → {A' → R^+}} as

$$L(x)(t)(b) = \begin{cases} T_{end}, \text{ if } x(t') \text{ does not include b for any } t', t \le t' < T_{end}, \\ \min\{t' \mid x(t') \text{ includes b at } t', t' > t\}, \text{ otherwise} \end{cases}$$

for any x ε X, t ε T, and b ε A'. L(x) is called a time-list representation of x.

The function L has its inverse which is written as L^{-1}. For a discrete event system S whose input space is X, the time system whose input space is L(X) is written as L(S).

Definition 10. *discrete event dynamical system [4]*

Let S be a discrete event system. A discrete event dynamical system of S is a state space representation of L(S).

Theorem 1

Let $<\phi, \Pi_F>$ be a three phase simulation system. Then $<\phi, \Pi_F>$ is a discrete event dynamical system of $L^{-1}(Res(<\phi, \Pi_F>))$.

Let $<\phi, \Pi_F>$ be a business transaction system. If every activty acts on files with preservation of relations on the file of $<\phi, \Pi_F>$ then the consistency condition on the file is preserved.

5. Conclusion

A state space representation of transaction system is proposed. Its state space is files and internally set times when some internal activities occur. This model provides a conceptual basis of both analysis and design of business transaction system.

It provides how a file system is modeled and used in a business system.

It also can be used to explain the reason why some of documents and activities in information systems methodologies are indispensable. Most of business analysis in information systems methodologies contain analysis of flows of physical objects and of information through functional units in an organization. If some of them has character that accumulates, then all of those must captured in modelling of dynamic behavior of business transaction, otherwise the behavior cannot be correctly reflect actual one. Since this task is usually undertaken by writing data flow diagrams or something like that, many methodologies have the diagrams as the part of the output documents. In other words, a collection of information requirements solely, without other information like analyzed flows, may not provide a sound model of the target system's behavior.

Acknowledgement

This research is partially supported by the University of Tsukuba Project Research.

References

[1] R. Hotaka: *Database Systems and Data Model*, Ohm, 1989 (in Japanese).

[2] M.D. Mesarovic, Y.Takahara: *Abstract Systems Theory*, (Lecture Notes in Control and Information Sciences 116), Springer, 1989.
[3] M. Pidd: *Computer Simulation in Management Science* (Second edition), John Wiley & Sons, 1984.
[4] R. Sato: *A Formulation of a Simulation Modelling Methodology*, Inst. Socio-Economic Planning Discussion Paper No.437, University of Tsukuba, 1990.