Chapter 5

Adaptability Evaluation of Software Systems

The software systems implemented in the above way are adaptable to the requirements, as far as we follow the procedure correctly. However, there could be errors made during the modeling or the software composition stage. In addition, we sometimes are not able to compose software systems arbitrarily due to restrictions of the components or the middlewares we use. For example, some components might not be able to run simultaneously, although our method concludes to execute them concurrently.

Therefore, we need to evaluate adaptability of the composed software systems. This chapter discuss two ways to evaluate adaptability of the implemented software systems to the original enterprise models.

One is a way to evaluate static adaptability, which ensures functional equivalency between a software system and an enterprise model. Σ homomorphism in Σ algebra is used as a measure of adaptability.

The other is a way to evaluate dynamic adaptability, which ensures behavioral equivalency between a software system and an enterprise model. Process algebra is used to compare the behavior of a software system with an enterprise model.

There could be another viewpoint of adaptability, which is defined as *data adaptability* in this thesis, or often is referred to as *data consistency* in database systems. Although this adaptability is important from practical viewpoints, it is a different concept of adaptability from those which are dealt with in this thesis. Therefore, it is not discussed in this chapter. A summarized discussion on this topic appears in Appendix A.

5.1 Adaptability of Software Systems

The adaptability of software systems can be defined as the degree that they satisfy the requirements.

There are many viewpoints of requirements in software systems. It depends on problem domains which aspects are of importance. For example, fault tolerance is an indispensable requisite for such systems as plant control and aviation control systems, and adaptability of these software systems should be evaluated from this viewpoint. Other viewpoints often referred to are performance, cost, usability, and so on [72]. However, the most essential and important viewpoints which are common among key problem domains of business applications are that:

- Each element in the software system performs a function or functions required from the business operations, and
- Behaviors of the software system reflect the business operations or the business processes.

This thesis discusses only the above viewpoints of adaptability, since we discussed enterprise modeling and software system construction according to the above viewpoints in the previous chapters. We refer to the former as *static adaptability* and the latter as *dynamic adaptability*.

When a software system satisfies static adaptability, it transforms data correctly every time they are put into it. Since this transformation is done by a module or a component in the system, static adaptability can be regarded as module or component level adaptability.

On the other hand, when a software system satisfies dynamic adaptability, operating the software system corresponds to the progress of the business process. In other words, the business process can proceed by operating the software system. This adaptability deals with the behavior of the software system, and can be regarded as system level adaptability.

In order to evaluate the adaptability of software systems to the requirements, we need to model the software systems to express functional and behavioral aspects concisely. Since the requirements were modeled in the form of CPN, it is desirable to model the software systems in the same form.

We can build software system models from the knowledge of them, as we did in enterprise modeling.

Roughly speaking, there are two ways to obtain the knowledge of software systems regarding their functionality and behavior. One is code inspection which analyzes the internal structure of the software systems. The other is external specification checking which examine the input/output relationships and the interfaces

of each component, along with the interrelationships among the components when they are on the production runs.

Since we focused on the external views of functionality and behavior in enterprise modeling in Chapter 3, external specification checking would be suitable to our purpose.

Today's software systems which support business processes are usually implemented as transaction processing systems [44]. Therefore, we focus on the transactional aspects of software systems in order to model them.

Functional and behavioral aspects of software systems can be depicted through the description of each atomic transaction (a transaction with ACID - Atomicity, Consistency, Isolation and Durability - properties) and their semantics (interdependency between transactions [1]) [25].

ACID properties are those which the smallest units of transaction processing must have, and are defined as follows.

1. Atomicity: All the operations of a transaction must be treated as a single unit, hence, either all the operations are executed, or none.

2. Consistency: If a transaction is executed alone, the transaction takes the database from one consistent state to another. When transactions are executed concurrently, the database management system must ensure the execution of a set of concurrent and correct transactions also maintains the consistency of the database.

3. Isolation: Each transaction must be able to observe the consistent database, i.e. not to read the intermediate result of other transactions.

4. Durability: The result of a committed transaction must be made permanent in the database in spite of failures.

From the knowledge of those atomic transactions and their interrelationships, we can make a software system CPN model in the following way.

1. Each transaction becomes a transition. Each function performed by the transaction becomes an arc expression function. Union of inputs to and outputs from the functions compose color sets.

2. Either an organization or a person that performs a transaction becomes an input place. A color function of a place denotes the data that the organization or the person have to deal with.

[1] Since we focused on execution orders of activities to express the behavior of business processes, we define the behavior of software systems as execution sequences of transactions

3. Structure of CPN, that is, connection between places and transitions, is determined from transaction semantics.

4. Each of guard and/or initialization functions is made from transaction semantics.

Figure 5.1 (CPN Diagram) and Table 5.1 (CPN Description Table) show an example of a software system CPN model, which is supposed to support the business process described in Section 3.2.

We can make the software system CPN model in the similar way to the business process CPN model. In order to make the CPN model, we need the knowledge on the inputs, outputs and the transformation rule of each transaction, along with the knowledge on the organization which performs the transaction.

In addition, we need such transaction semantics that :

1. "Inventory Check", "Availability Check" and "Credit Check" transactions must precede "Evaluation" transaction

2. Before performing "Rejection Letter" transaction, "Rejection Record" transaction must be accomplished

This CPN model represents both the functional aspect and the behavioral aspect of the software system in the same way as the business process CPN model discussed in Section 3.2.

In many cases, we have to deal with multiple atomic transactions as one logical unit to be atomic. In order to handle such a set of transactions, we need to enhance the transaction model (ACID transaction model). This problem is discussed in Appendix A.

5.2 Evaluating Adaptability from Static Viewpoint

A CPN model discussed in the previous sections has two aspects. One is static and the other is dynamic. A static aspect is often referred to as functionality, while a dynamic one is referred to as behavior.

In this section, we deal with evaluating static adaptability of software systems. In order to make evaluation simple, it is desirable to express only a static part of a CPN model. Σ algebra could be a candidate for this purpose, since the functionality of a CPN model can be expressed as Σ algebra.

Moreover, static adaptability we define means that each function performed in a business process corresponds to a function in a software system. This correspondence can be expressed as Σ homomorphism between Σ algebras.

Table 5.1: Software system CPN model description

Color sets

C_1 : product name, C_2 : product number
C_3 : quantity, C_4 : customer name
C_5 : credit number, C_6 : customer address
C_7 : check result, C_8 : order number
C_9 : warehouse number, C_{10} : price ...
$D_1 : C_1 \times C_3 \times C_4 \times C_5 \times C_6$, $D_2 : C_8 \times C_2 \times C_3$
$D_3 : C_8 \times C_2 \times C_3 \times C_7$, $D_4 : C_8 \times C_4 \times C_5 \times C_6$
$D_5 : C_8 \times C_4 \times C_5 \times C_6 \times C_7$
...

Initial Marking (Initialization function)

$M(p_1) = \langle x_1, x_2, x_3, x_4, x_5 \rangle \in D_1$

Places

$P = \{p_1, p_2, \ldots, p_{13}\}$

Transitions

$T = \{t_1, t_2, \ldots, t_{11}\}$

Color function

$C(p_1) = D_1, C(p_2) = D_2, C(p_3) = D_4$, $C(p_4) = D_3 \cup D_5$,
...

Arc Expression Function

$E(p_1 \rightarrow t_1) = Id, E(p_2 \rightarrow t_2) = Id, \ldots$
Id : Identity function
$E(t_1 \rightarrow p_2) = (h_1(x_1), h_2(x_2), Proj\ 2)$
h_1 : assigns an *order number*
h_2 : transforms a *product name* to a *product number*
$Proj\ i$: Projection
$E(t_1 \rightarrow p_3) = (h_1(x_1), h_2(x_2), Proj3, Proj4, Proj5)$
$E(t_2 \rightarrow p_4) = if\ x_2 \in X_1 \subseteq C_2$ (the product is available)
then $1`(Proj1, Proj2, Proj3, \top, h_3(x_2))$
else $1`(Proj1, Proj2, \bot)$
$h_3 : C_2 \rightarrow C_9$ assign warehouse number
...

Initialization function

$I(p_1) = 1`(x_1, x_2, x_3, x_4, x_5) \in D_1$
$I(p) = \emptyset\ (p \neq p_1)$

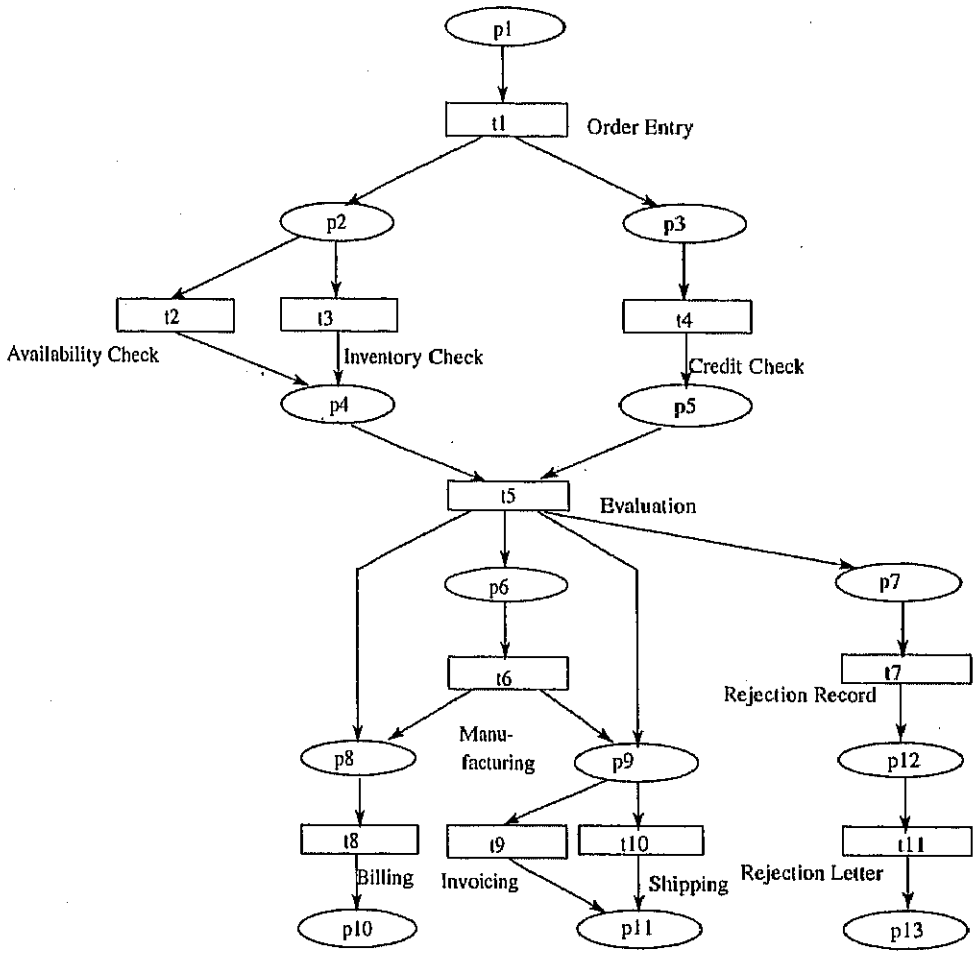

Figure 5.1: Sample software system

Therefore, by transforming CPN models to Σ algebras, we can evaluate the static adaptability of software systems.

As described in section 4.1.1, CPN models which are constructed from software systems can be transformed into Σ algebras.

A Σ algebra derived from a transition t or a set of transitions $\{t_i\}$ is denoted by $\mathcal{A}_t$ or $\mathcal{A}_{\{t_i\}}$ respectively in this thesis.

The adaptability of a software system to a business process can be evaluated by Σ homomorphism between two Σ algebras derived from those realms in the following way.

As stated in section 4.1, Σ homomorphism $\eta = \{\eta_\sigma | \sigma \in S, \eta_\sigma : A_\sigma \rightarrow B_\sigma\}$ is a family of functions such that

$\forall f \in \Omega_{\sigma_1 \ldots \sigma_n, \sigma}, \forall a_i \in A_{\sigma_i}$

$\eta_\sigma(f_A(a_1, \ldots, a_n)) = f_B(\eta_{\sigma_1}(a_1), \ldots, \eta_{\sigma_n}(a_n))$

$(n \geq 0)$

where $\mathcal{A}$=(A, F) and $\mathcal{B}$=(B, G) are Σ algebras. f_A and f_B are the functions in F and G respectively, or elements of A and B if $n = 0$.

Let $\mathcal{A}$=(A, F) and $\mathcal{B}$=(B, G) be Σ algebras derived from an enterprise model and software system model respectively. Those η_σs can act as interpreters between the business realm and the software realm. A transformation by a function f_A in an enterprise model can always be mapped to a transformation by a function f_B, and the transformation by f_B can be interpreted as the transformation by f_A in the enterprise model. Each transformed values are correlated by Σ homomorphism η_σ.

When some part of the software system is adaptable to some part of the requirement model, we define such adaptability as *partial adaptability*. In order to handle this partial adaptability of software systems, we define *subnets* of CPN models. A *subnet* of a CPN model is a set of interconnected [2] transitions, places connected to the transitions and arcs between them within the model. Since a subnet of a CPN model is also a CPN model, we can derive a Σ algebra from it.

In this thesis, $SN(\{t_i\})$ denotes a subnet with a set of transitions $\{t_i\}$. In order to distinguish subnets in business process models from those in software system models, we use $SN_B(\{t_i\})$ for a business process model, and $SN_S(\{t_i\})$ for a software system model respectively.

A subnet of a business process model means a part of the business process composed of interrelated activities, while a subnet of a software system model means a part of the software system composed of interrelated transactions.

When there is a Σ homomorphism from a business process subnet to a software system subnet, each function in the former subnet has the equivalent function in the latter subnet. This means the business process subnet is equivalent to the software system subnet, in other words, the part of software system is adaptable to the part of business process.

Since Σ homomorphism assumes two Σ algebras are derived from a common signature $\Sigma = (S, \Omega)$, we must define the common signature between Σ algebras from a business process and a software system.

We can build up a common set of sorts S by defining the correspondences between carrier sets in each Σ algebra. Afterwards, a common set of operation names Ω is built up by defining the correspondences between functions in each Σ algebra. The corresponding functions must have the same arity and result sort. Since we can define arbitrary correspondence, there could be multiple common signatures.

If there are carrier sets or functions with no correspondence, they should be

[2] In a CPN graph, if there is only one place between two transitions, we call these transitions are interconnected

excluded from adaptability evaluation. Therefore, the only subnets composed of the common signature are to be evaluated.

The evaluation of static equivalency can be done by performing the following steps:

1. Select one transition t_i in the business process CPN model, then construct the Σ algebra $\mathcal{A}_{t_i}$ using the procedure described in Section 4.1.1.

2. Find a set of transitions $\{t'_j\}$ in the software system CPN model, from which we can derive the Σ algebra $\mathcal{A}_{\{t'_j\}}$ that has a Σ homomorphism from the algebra in step 1 (Multiple sets of transitions could be found in the software system CPN model). If a Σ homomorphism is found, mark the transition t_i.

3. Repeat above steps over all of the transitions in the business process CPN model.

4. For each set of interconnected transitions $\{t_i\}$ in the business process model, which were marked in step 2, construct Σ algebra $\mathcal{A}_{\{t_i\}}$.

5. Examine existence of subnets in the software system CPN model, which derive a Σ algebra that has Σ homomorphism from $\mathcal{A}_{\{t_i\}}$ in step 4.

By this procedure, we can find the subnets in the business process model which have the corresponding equivalent subnets in the software system model.

We can identify the largest equivalent subnets in the enterprise model and the software system model, by making the union of all the equivalent subnets in each model. If there are unequivalent subnets in the enterprise model, we have to develop appropriate components correspond to the functionality of such subnets.

5.3 Evaluating Adaptability from Dynamic Viewpoint

Static adaptability discussed in the previous section assures that each activity in the business process can be performed by a transaction in the software systems. In this section we discuss another aspect of adaptability.

A business process is composed of ordered activities, and is recognized as behavior. A software system also has behavioral aspect, since it is designed to support a specific business process. This property of a software system is called *flow dependency* [46], which means that each program can be executed in a particular sequence.

In order to describe and analyze this dynamic or behavioral aspect of models, and to evaluate equivalency between two models (i.e. a business process and a software system), we use process algebra for a rigorous discussion.

Process algebra is an algebraic approach for dealing with nondeterministic and concurrent transition systems or processes. Several different approaches are proposed in process algebra, they are CCS [49],[50], CSP [38] and ACP [3], [5].

Those approaches have their unique characteristics, e.g. CSP can handle the communication among more than two agents while the others can not, ACP has rigorous axiomatic definition of process and their expressions while the others have more intuitive ones, and CCS has many kinds of equivalency concepts between two processes while the others have much fewer.

Since we need flexible definitions of behavioral equivalency [3] in order to evaluate dynamic adaptability, CCS is adopted in this thesis.

5.3.1 Transforming Two Models into CCS

In CCS, processes are described using the syntax and the semantics in Table 5.2. CCS expressions and Petri nets are deeply related each other [29]. In case of ordinary Petri nets, actions in CCS expressions correspond to transactions in Petri nets. For example, CCS expressions $P = a|b$ and $Q = a.b + b.a$ have the corresponding Petri nets (ordinary Petri nets) $N(P)$ and $N(Q)$ as shown in Figure 5.2. However, in case of CPN, they are arc functions that correspond to actions, since a transition could perform different functions according to the input colors to the transition.

We assume that CPN models in this thesis become disabled after finite number of transition firings take place. Therefore, if there is a loop in CPN model like Figure 5.3, the iteration of firing sequence $t_2 \rightarrow t_3$ must end within finite firing counts. This assumption is realistic for business process CPN models and software system CPN models, since business processes or software systems which are running eternally must not exist in enterprise back-office applications we focus on in this thesis.

In addition, we also assume that all initial markings are only in source places, that is, places without input transitions.

When a CPN model has no loops, it can be expressed by CCS. The following procedure can be used for the purpose:

1. Divide the CPN model into two parts. One consists of the places that have initial marking and the transitions that are connected to those places. The rest part of the CPN model is denoted by the *clouded* P1, and its CCS expression is E(P1). (Figure 5.4)

[3] such as dynamically adaptable software systems by using glue codes

Table 5.2: CCS syntax and semantics

CCS Syntax
$\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ ($\mathcal{A}$: set of names $\overline{\mathcal{A}}$: set of conames) $\mathcal{A}ct = \mathcal{L} \cup \{\tau\}$ (τ : invisible action) $\mathcal{X}$: a set of process variables $\mathcal{K}$: a set of process constants $\mathcal{E}$: a set of process expressions $\mathcal{E}$ includes $\mathcal{X}$ and $\mathcal{K}$ and contains the following expressions, where E and E_i are already in $\mathcal{E}$ $\alpha.E$ a Prefix $\sum_{k\in I} E_i$ a Summation (if $I = \{0, 1\}$ then $E_1 + E_2$) $E_1 \mid E_2$ a Composition $E \setminus L$ a Restriction $E[f]$ a Relabelling (f is a relabelling function)
CCS Semantics
ACT : $\frac{}{\alpha.E \xrightarrow{\alpha} E}$ SUM$_j$: $\frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{k\in I} E_i \xrightarrow{\alpha} E'_j}(j \in I)$ COM$_1$: $\frac{E \xrightarrow{\alpha} E'}{E\mid F \xrightarrow{\alpha} E'\mid F}$ COM$_2$: $\frac{F \xrightarrow{\alpha} F'}{E\mid F \xrightarrow{\alpha} E\mid F'}$ COM$_3$: $\frac{E \xrightarrow{l} E' \quad F \xrightarrow{\bar{l}} F'}{E\mid F \xrightarrow{\tau} E'\mid F'}$ RES : $\frac{E \xrightarrow{\alpha} E'}{E\setminus L \xrightarrow{\alpha} E'\setminus L}(\alpha, \bar{\alpha} \notin L)$ REL : $\frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$ CON : $\frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}(A \stackrel{def}{=} P)$ $P \xrightarrow{\alpha} P'$ means a transition from state P to state P' by action α.

2. As the result of the procedure described in section 4.1, a number of S-sorted functions are defined from each output arc function. Let an arc function be f_i, and the S-sorted functions extracted from it be $g_{ij}(j = 1, \ldots, n_i)$. An action from S-sorted function g_{ij} is denoted by a_{ij}.

3. For one output arc function f_i, CCS expression $(a_{i1} + a_{i2} + \ldots a_{in_i}).E(P1)$ is derived

4. For concurrency of output arcs (like f_9 and f_{10} in Figure 5.4), CCS expression $((a_{i1}|a_{j1}) + \ldots + (a_{in}|a_{jn})).E(P1)$ is derived. [4]

[4]The same number of S-sorted functions are derived from output arc functions, if they are from the same transition.

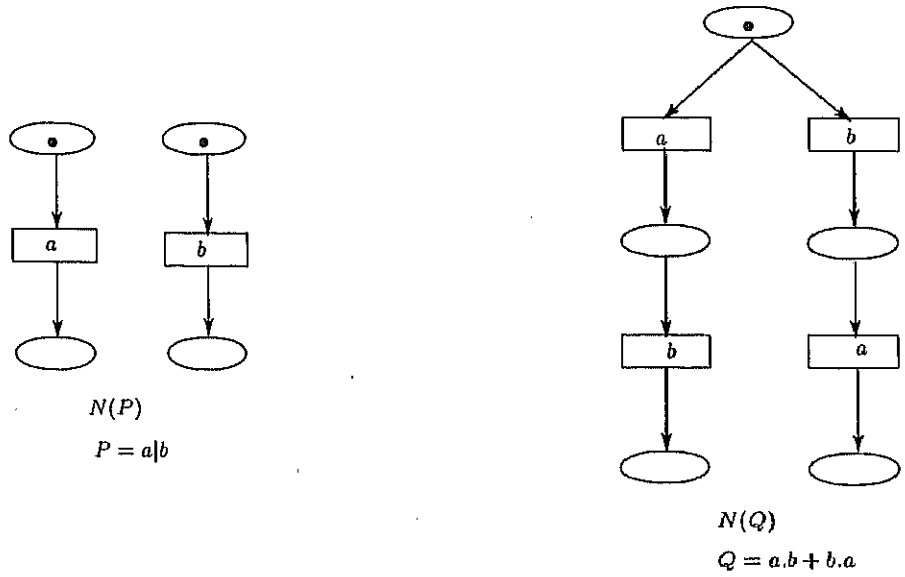

Figure 5.2: Petri nets and CCS

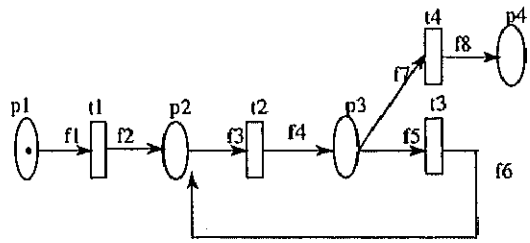

Figure 5.3: A loop in a CPN model

5. For conflict of transitions t_i and t_j (like t1 and t2, or t3 and t4), CCS expression $(E(t_i) + E(t_j)).E(P1)$ is derived, where $E(t_i)$ is the CCS expression derived from output arc functions from transition t_i.

6. apply the above procedure to CPN model P1.

Since we assume there are no loops within the CPN model, the above procedure ends within finite number of iteration. The *clouded* area P_1 becomes smaller as the procedure proceeds, and finally disappeared.

When loops are included in a CPN model, those loops must exit within finite number of iteration, otherwise the business process or software system that is represented by the model will not end forever, and such a business process or

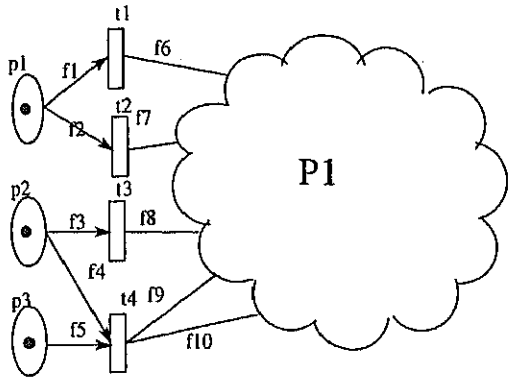

Figure 5.4: A CCS expression from a CPN model

software system is not realistic.

A loop within CPN model is repetition of a series of transitions like $t_1, t_2, t_3, t_1, t_2, t_3, \ldots$.

The CCS expression P of this loop can be denoted by

$$P = (E(t_1).E(t_2).\ldots.E(t_n)).P$$

Since the loop is finite, this recursive notation is unfolded as

$$P = (E(t_1).\ldots.E(t_n)).(E(t_1).\ldots.E(t_n)).$$
$$\ldots.(E(t_1).\ldots.E(t_n))$$

For example, we get the CCS expression of a CPN model including a loop as shown in Figure 5.3 :

$$E(t_1).(E(t_2).E(t_3).(E(t_2).E(t_3).(E(t_2).E(t_4).0)+$$
$$E(t_2).E(t_4).0) + E(t_2).E(t_4).0)$$

by performing the previous procedure. In this example, we assume the maximum loop count is two.

5.3.2 Equivalency Concepts in CCS

Since the purpose of dynamic adaptability evaluation is to determine whether two systems (a requirement model and a software system model) act equivalently in some sense, we discuss the equivalency concepts defined in CCS in this section. CCS has various concepts of equivalency between two processes. There are six major concepts of equivalency, that is:

1. Strong equivalence which focuses on actions eligible to be taken, and two processes are regarded as equivalent when all the eligible actions are the same at any point.

2. Weak equivalence which is similar to strong equivalence, however each process can include *invisible actions* between actions.

3. Observation congruence which is similar to weak equivalence excepting the usage of *invisible actions*.

4. Testing equivalence which focuses on a set of actions that makes a process succeed. Two processes are regarded as equivalent when they have the same such set.

5. Failure equivalence which focuses on a set of actions that makes a process fall into *deadlock*. Two processes are regarded as equivalent when they have the same such set.

6. Trace equivalence which focuses on all the sequences of actions that are eligible to be taken. Two processes are regarded as equivalent when those sequences are the same between them.

We only consider 1 (strong equivalence), 2 (weak equivalence) and 3 (observation congruence), since 4 (testing equivalence) and 5 (failure equivalence) do not deal with the execution of each action, and are not suitable to handling the behavioral aspects of the models, while 6 (trace equivalence) is too restricted concept to handle the behavior of the models.

The formal definitions of 1, 2 and 3 are shown in Table 5.3. In the following sections, we discuss how these concepts are used for evaluating adaptability of a software system to a business process. Only the statically equivalent subnets are evaluated, since static equivalency is obviously prerequisite to dynamic equivalency.

Strong equivalence is used to evaluate the strict equivalency between two systems, while weak equivalency and observation congruence are considered to evaluate the equivalency under more slack conditions.

5.3.3 Evaluating Dynamic Adaptability with Strong Equivalence

As described in Section 3.1, the behavior of an enterprise is defined as ordered sequences of tasks in this thesis. Those tasks are transformed into transitions in a CPN model, then are embedded in S-sorted functions of Σ algebra derived from the CPN model. Similarly, the behavior of a software system is defined as ordered sequences of transactions as discussed in Section 4.1. Those transactions are transformed into transitions in a CPN model, then imbedded in S-sorted functions of Σ algebra derived from the CPN model.

As mentioned in Section 5.3.2, we only deal with the statically equivalent subnets in CPN models, therefore each S-sorted function in Σ algebra of an en-

Table 5.3: Equivalency between CCS expressions

Strong Bisimulation and Strong Equivalence
Let P, Q be processes, a binary relation $S \subseteq P \times P$ implies, for all $\alpha \in Act$ $\forall\alpha(P \xrightarrow{\alpha} P')\ \exists Q'(Q \xrightarrow{\alpha} Q'$ and $\langle P', Q'\rangle \in S)$ $\forall\alpha(Q \xrightarrow{\alpha} Q')\ \exists P'(P \xrightarrow{\alpha} P'$ and $\langle P', Q'\rangle \in S)$ $\sim = \cup$ S : S is strong bisimulation P and Q is strongly equivalent if $P \sim Q$.
Bisimulation and Weakly Equivalence
Let P, Q be processes, a binary relation $S \subseteq P \times P$ implies, for all $\alpha \in Act$ $\forall\alpha(P \stackrel{\alpha}{\Longrightarrow} P')\ \exists Q'(Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ and $\langle P', Q'\rangle \in S)$ $\forall\alpha(Q \stackrel{\alpha}{\Longrightarrow} Q')\ \exists P'(P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ and $\langle P', Q'\rangle \in S)$ $\approx = \cup$ S : S is bisimulation P and Q is weakly equivalent if $P \approx Q$ where $P \stackrel{\alpha}{\Longrightarrow} P'$ means $P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* P'$ $t = \alpha_1 \ldots, \alpha_n$ $\hat{t} \in (Act)^*$ a string made from t, removing all "τ"s $\stackrel{t}{\Longrightarrow} \stackrel{def}{=} \xrightarrow{\alpha_1} (\tau)^* \xrightarrow{\alpha_2} \ldots (\tau)^* \xrightarrow{\alpha_n} (\tau)^*$
Observation Congruence
P and Q are observationally congruent, if $\forall\alpha(P \stackrel{\alpha}{\Longrightarrow} P')\ \exists Q'(Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \approx Q')$ $\forall\alpha(Q \stackrel{\alpha}{\Longrightarrow} Q')$ then $\exists P'(P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \approx Q')$ we write $P \cong Q$ when P and Q are observationally congeuent.

terprise CPN model has the corresponding S-sorted function in Σ algebra of a software system CPN model.

Therefore, by regarding those S-sorted functions as the same action, we can define the behavioral adaptability of software systems to the requirements in the following way.

1. When a task is performed in an enterprise model, that is, an S-sorted function in which the task is imbedded is performed, the corresponding S-sorted function in a software system model is eligible to be performed.

2. When a transaction is performed in a software system model, that is, an S-sorted function in which the transaction is imbedded is performed, the

corresponding S-sorted function in an enterprise model is eligible to be performed.

The above two conditions assure the set of ordered sequences of tasks in an enterprise model and the set of those of transactions can be mapped into the same set of ordered sequences of actions in terms of CCS.

There could be multiple statically equivalent subnets within a business process model and a software system model. Let them be $\{SN_{Bi}\}$ and $\{SN_{Si}\}$, where $i = 1, \ldots, n$, and the CCS expression from $\{SN_{Bi}\}$ be P_i and that from $\{SN_{Si}\}$ be Q_i.

These CCS expressions P_i and Q_i are a part of the CCS expressions of the whole business and software system models. Let E_B and E_S be CCS expression of a business process and a software system respectively.

Since P_i and Q_i are a part of E_B and E_S respectively, E_B and E_S can be denoted by $E_B(\ldots, P_i, \ldots)$ and $E_S(\ldots, Q_i, \ldots)$.

If we can prove $P_i \sim Q_i$ then we can conclude

$E_B(\ldots, P_i, \ldots) \sim E_B(\ldots, Q_i, \ldots)$

from the property of strong bisimulation [50].

This means that subnet SN_{Bi} and SN_{Si} are dynamically equivalent. The subnet named SN_{Bi} in the business process can be replaced or performed by the subnet named SN_{Si} in the software system.

Several methods are available for proving strong equivalency, e.g. a method using axiom system for strong bisimulation or using logic for specification [50].

All models we deal with in this thesis are finite, that is, all CCS expressions terminate after performing finite number of actions. In such cases, we can use more simple method. This method examines all possible state transitions in CCS expression.

For given CCS expressions P_0 and Q_0, which are from a business process subnet and a software system subnet that are statically equivalent, we can examine equivalency between them by the following procedure.

1. Let $(P_i)_k$ and $(Q_i)_k$ be one of the processes which are the result of performing i actions from P_0 and Q_0, $(a_i)_{kj}$ and $(b_i)_{kj}$ be one of the actions that can be performed from the process $(P_i)_k$ and $(Q_i)_k$ respectively.

2. For any $(P_i)_k$, if $(Q_i)'_k$ exists, and $\{(a_i)_{kj}\} = \{(b_i)_{k'j}\}$, then there is a possibility of $P_0 \sim Q_0$.

3. Otherwise, $P_0 \not\sim Q_0$.

4. If there is a possibility of $P_0 \sim Q_0$, then compose a relation.

   $R = \{(P_i)_k, (Q_i)'_k\}$. If $(P_i)_k$ has multiple partners satisfying step 2, make up all possible relations.

5. Examine whether the above relation or relations are strong bisimulation according to the definition in Table 5.3.

By repeating above procedure over all statically equivalent subnets, we get the subnets which are statically (or functionally) and dynamically (or behaviorally) equivalent.

5.3.4 Evaluating Dynamic Adaptability with Weak Equivalence and Observation Congruence

In this section, we discuss a way for evaluating dynamic adaptability of one subnet to the other, which are not strongly equivalent.

When two CCS expressions (or two processes represented by those expressions) are not strongly equivalent, there are some reachable states by performing the same sequence of actions, at which the two processes have the different sets of actions eligible to be taken.

From practical viewpoint, it seems possible to perform a non-eligible action by shifting the state by adding another action before the non-eligible action.

In a software system, such situation is possible when a transaction is not able to be performed by the lack of preconditional transactions which happen to be excluded from the software system.

For this purpose, we consider weak equivalency and observation congruence instead of strong equivalency. Weak equivalence and Observation congruence are the enhanced concepts of strong equivalence. They allow processes to perform any number of invisible action "τ" when evaluating equivalency between two processes. They are very similar concepts and either of them seems suitable for equivalency evaluation.

However, when two processes P_i and Q_i are weakly equivalent, that is, $P_i \approx Q_i$, this does *NOT* imply $R + P_i \approx R + Q_i$, where R is any process. Therefore, unlike strong equivalence, we cannot conclude that

$E_B(\ldots, P_i, \ldots) \approx E_B(\ldots, Q_i, \ldots)$ if $P_i \approx Q_i$.

Observation congruence relieves this undesirable property, since we can conclude that

$E_B(\ldots, P_i, \ldots) \cong E_B(\ldots, Q_i, \ldots)$ if $P_i \cong Q_i$.

This means a subnet denoted by P_i in a business process can be performed by a subnet denoted by Q_i in a software system. This is similar to a property of strong equivalency described in the Section 5.3.3.

The following procedure shows the steps we follow in order to find observation congruence between a business process subnet P_0 and software system subnet Q_0, which are statically equivalent but not strongly equivalent. (Symbols and notation used here are the same as those used in the Section 5.3.3.)

1. Find a pair of $(P_i)_k$ and $(Q_i)_{k'}$ which satisfies
   $\{(a_i)_{kj}\} \supset \{(b_i)_{k'j'}\}$ $(\{(a_i)_{kj}\} \neq \{(b_i)_{k'j'}\})$.
2. Search a set of processes $\{(Q_{i+1})_l\}$
   which satisfies
   $\bigcup_p \{(b_{i+1})_{lp}\} = \{(a_i)_{kj}\} - \{(b_i)_{k'j'}\}$.
3. If above set $\{(Q_{i+1})_l\}$ is not found, P_0 and Q_0 are not observationally congruent.
4. If above set $\{(Q_{i+1})_l\}$ is found, draw arrows labeled "τ" from $(Q_i)_{k'}$ to each $(Q_{i+1})_l$.
5. Compose a relation $R = \{(P_i)_k, (Q_i)_{k'}\}$. If $(P_i)_k$ has a multiple partners, make up all possible relations.
6. Examine whether the above relation or relations are observationally congruent according to the definition in Table 5.3.

The rest of this work is to design and implement each "τ" as hidden glue codes.

Software systems discussed in this thesis are those for enterprise back-office applications as mentioned above, most of them are usually executed under the control of "database and transaction management system". In such case there is a practical issue known as *data consistency*. This issue can be regarded as a kind of adaptability of software systems, however it is a little aside from the main subjects of this thesis. Therefore only a brief discussion is shown in Appendix A.