QUALITATIVE BEHAVIOR MODELING OF
INFORMATION PROCESSING COMPONENTS

Sang Hun Oh

Industrial Electronics R&D Center
Hyundai Electronics Industries Co., Ltd.
San 136-1, Ami-ri, Bubal-eub, Ichon-kun
Kyoungki-do, 467-860, KOREA

Yoon Joon Lee and Myoung Ho Kim

Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1, KuSung-dong, YuSung-gu
Taejon, 305-701, KOREA
E-mail: mhkim@cs.kaist.ac.kr, Fax: +82-42-869-3510

ABSTRACT

This paper presents qualitative behavior modeling of Information Processing Components (IPCs), which are domain-specific software components for modeling reusable functions in database management systems (DBMSs). After introducing a detailed formal syntax of IPCs, we propose their modular, qualitative behavior generation mechanism. The proposed mechanism is based on qualitative modeling and simulation, thus produces a sequence of temporal state transitions of modeled IPCs. The proposed modeling mechanism aims to facilitate the scientific measurement of DBMSs.

1. Introduction

Besides the scientific measurement environment, it is also important to make use of systems domain knowledge when measuring software systems[2, 4, 10]. Software Quality Manager, which was proposed in [6], is a knowledge-based software metrics management system that provide software engineers with collective and comprehensive measurement environment that provides software engineers with domain knowledge about application systems and software measurement in an engineering way.

In the Software Quality Manager, Information Processing Components (IPCs) were designed to support both testing and measurement of DBMSs simultaneously through systematic management of the ever-increasing amount of knowledge about their software products[7]. As the IPCs were developed according to domain analysis technique[9], They can also aid in the construction activity by providing designers with development knowledge in the reusable forms. They can support testing job by generating their temporal behavior from the information about their interconnection with other components and their operational characteristics.

In this paper, a qualitative behavior modeling method of IPCs is proposed. The proposed method is based on qualitative modeling and simulation[3], thus produces a sequence of temporal state transitions of modeled IPCs, as a behavior of modeled DBMSs. As the complexity of software systems is so high that there must be a way to reduce the complexity in a modular way. To deal with such a high complexity, the proposed method also adopts software component concept, which is realized by IPCs. This paper describes how to generate the behavior of IPCs, especially on the qualitative modeling aspect. The proposed method aims to facilitate the scientific measurement of DBMSs. The produced behavioral information can be used by those who perform software measurement.

This paper is organized as follows. Section 2 presents the formal structure of IPCs. Then behavior generation method of DBMSs using IPCs is proposed in section 3. The application areas of the generated behavior include both assessment and measurement activities. Finally, concluding remarks are presented in section 4.

2. Design of Information Processing Components

2.1 Support of behavior analysis of designed DBMSs

Functional prototyping and simulation can reduce the difficulty of dynamic analysis of software systems being developed[2, 8]. To support the testing, we aim at utilizing the functional knowledge about DBMSs for the qualitative simulation of the dynamic behavior of designed DBMSs. If the functional knowledge is represented in independent objects, it is rather simple to generate the behavior of designed DBMSs. As a result, IPCs are determined to organize the functional knowledge of DBMSs in an object-oriented fashion. The reason of adapting object-oriented concept is that objects' external behavior is represented explicitly by their message transmission, and their internal behavior is hidden from other objects. Such operational characteristics make it simple to observe their interactions at the component and the system level.

We also must adopt the finite state machine as a behavior generation mechanism of IPCs. In finite state machine paradigm, a behavior of IPCs is represented by a sequence of temporal state changes. Thus, IPCs must be equipped with their primitive operations and their causality information, together with their meaningful states. As IPCs are also based on the concept of objects, message passing mechanism can be used to represent the interaction between IPCs. The behavior generation of IPCs can contribute to the human beings' enhanced understanding of the functional analysis of the system being developed. This is because the execution history
of DBMSs can become visible forms.

2.2 Support of DBMS measurement

The generated behavior of IPCs is represented by a sequence of state transition operations. If we can incorporate duration information into those operations in the sequence, it is possible to estimate total execution time of designed DBMSs. Such a work corresponds to a performance evaluation task. As the functions in IPCs are becoming stabilized, approximate performance parameters can be stored somewhere in the library. Besides performance metrics, other software metrics can be measured from the generated behavior of IPCs[8]. This idea needs more work, but we argue that it is possible when DBMSs are developed with IPCs.

2.3 Formal specification of IPCs

2.3.1 The name part

The name of a reusable function is stored in the name part of the generic structure. The name stored in the name part play the role of identifier of IPCs so that the corresponding IPC is uniquely identifiable in a full collection of information processing components. If there are many variants of an IPC, the name of individual variants is described by appending sequence number into the name of the IPC.

Another important factor, besides construction of reusable software components, to the success of software reuse is how to manage a collection of reusable software components and how efficiently the retrieval of them is performed. These factors are greatly influenced by the classification scheme of a library of reusable software components[1, 5]. The classification scheme amounts to abstraction of the library.

When IPC constructors determine that a certain function X has high reusability, the name of the function, X, is stored in the name part of corresponding IPC construct. The syntax of the name part is shown in Figure 1(b), which defines that the name of reusable functions consists of alphanumeric characters. Names of meaningful entities such as buffer, memory, storage structure, query processor and disks and of their related operations are candidates of possible IPCs.

The examples of the names of buffer-management IPC include buffer_initialization, buffer_operation, buffer_finalization, buffer_allocation, buffer_discard, buffer_get, buffer_flush, buffer_finalization, buffer_validate, buffer_read, buffer_write and so on. These IPCs can be grouped with respect to the characteristics of their operation. As shown in Figure 2, the grouping is based on the phase when each IPC can be able to be executed during the entire life cycle of buffer. Such a grouping is helpful to find out the causality between IPCs and to generate meaningful sequence of IPC invocations. That can also contribute to the placement of IPCs in the library. That is, the grouping criterion is a classification scheme of a library of IPCs.

<ipc_name> ::= <name>
<name> ::= <characters> { a meaningful list of characters }
<characters> ::= Al B C I I Y I Z I I I I I I I I I I I I I I I I I I I I I I I I y y y

(a) Syntax of information processing components

2.3.2 The teleology part

The teleology part contains the purpose of an IPC. The purpose describes developer's decisions and intention such as why this IPC is selected, i.e., the selection conditions. The teleology part can be said an abstracted description of the name part. Thus, it is possible to understand or estimate the functional details and objectives of an IPC. The purpose of an entire system can also be understood or estimated from the collection of the teleology parts of selected IPCs. This part is an accomplishment of the design principle that IPCs should be equipped with abstract symbols.

As developers generally describe their intention in a narrative form, it is very difficult to represent it systematically. The optimal approach to modeling of developers' intentions is for them to describe it concisely. Owing to the narrativeness, Estimation the behavior of an IPC from the teleology part is now not automatically performed.

<teleology_part> ::= { a description of purpose of an IPC }

Some examples of the teleology part are as follows.

1. buffer_read - read a page from disk to buffer.
2. buffer_flush - write all pages in buffer pool to disk.
3. buffer_allocanon - allocate a buffer to a page in a file.
4. buffer_discard - discard a page stored in a buffer.

2.3.3 The structure part

The structure part represents the interaction, inner structural and operational information of a function stored in an IPC. It includes internal interface description and private data structure of an IPC is stored in this part. The input/output interface is described by generic parameterization technique. DBMSs engineers can predict more correct behavior of the IPC by examining the structure part and the teleology part, but they cannot judge whether the predicted behavior is correct and complete. Other information stored in causality and behavior parts is essential to extract correct behavior of an IPC.

The structure part, which is divided into IMPORT, EXPORT and CONSIST_OF sub-parts, describes the interaction among IPCs and its environments. As it is simple and better to simplify the interaction complexity, we adopt import/export mechanism as paths to and from its environments. The IMPORT sub-part represents other IPCs referenced by the IPC. In specific, as
functions and data structures can become IPCs, we divided the imported IPCs into FUNCTION and DATA STRUCTURE types. The FUNCTION type includes IPCs which contain reusable functions; whereas DATA STRUCTURE type contains IPCs for meaningful, important abstract data types. Whereas, the EXPORT sub-part indicates IPCs provided by the IPC. The refined syntax of EXPORT sub-part is identical to those of IMPORT sub-part. The CONSIST_OF sub-part is to represent the interconnection information about sub-IPC's located in an IPC. It is a frequent case that a complex IPC is composed by several simpler IPCs. In this case, it is necessary to represent them and the relationships among sub-IPCs. Value NONE is specified in the <import-list> and <export-list> if no IPC is related.

If all of the sub-IPCs are stored in the IPC, that may be so complex that developers cannot understand the upper-level IPC. To avoid this problem, we determine a policy that no IPC can have detailed information about their sub-IPCs. In other words, the representation of sub-IPCs are stored in other independent IPCs and the upper-level IPC only references the sub-IPC's by holding their relationships in the CONSIST_OF and CAUSALITY part. Their CONSIST_OF part contains only the names of the related sub-IPCs; and their CAUSALITY part stores the interconnection information. Examples of the structure part for buffer_read is shown in Figure 3. Formal syntax of the structure part is depicted in Figure 4.

```
IPC buffer_read
  TELEOLOGY read a page from disk to buffer.
  STRUCTURE
    IMPORT FUNCTION
    IMPORT DATA STRUCTURE disk, bufferpool, page
    EXPORT FUNCTION buffer_read
    EXPORT DATA STRUCTURE: NONE
    CONSIST_OF disk_interface, page_reader

END_OF_IPC buffer_read
```

### Figure 3 Example of structure part for buffer_read IPC

#### 2.3.4 The causality part

The objectives of the causality part are two-folded. First, as DBMSs have a layered hierarchical architecture, it is not unusual that complex functions are composed of more than one IPCs. For the purpose of representational homogeneity between a DBMS being constructed and its selected IPCs, the DBMS should be regarded as a complex IPC. Thus, we must provide a mechanism to represent the nesting of IPCs in a uniform way.

Another purpose of IPCs is to generate information about the dynamic part of DBMSs. To this end, we must provide a mechanism to represent temporal execution sequence among IPCs in order to describe the behavior of DBMSs and IPCs. Such sequence information is stored in the causality part. The behavior generation mechanism is also useful to their scientific measurement. If the timing information of operations, which is modeled as a time interval in a certain state, can be added to the sequence information, it is possible to estimate the performance metrics of DBMSs.

An IPC needs scheduling mechanism of the invocation sequence of sub-IPCs, in order to perform its function correctly. The causality part contains such scheduling information so that developers can describe it conveniently and that testing of their correctness and completeness can be performed in a modular fashion.

First, it is necessary to identify the temporal operators for modeling the precedence condition between two sub-IPCs. For example, if sub-IPC A must be executed before sub-IPC B, then the sequence information is represented as "sub-IPC A precedes sub-IPC B," or "sub-IPC B follows sub-IPC A." Furthermore, we provide more semantically temporal operators to perform more deep semantic analysis on IPCs. These include ACCESSES, MODIFIES, NEEDS_VALUE_OF, SENDS_INFORMATION_TO, CREATES and ACTIVATES. As the meaning of each operator can be identified easily, we omit their more detailed explanation. If other semantic operators are identified, they can be added for more rich analysis and testing of IPCs.

```
<causality_part> ::= <uni_temporal_operator>
  <causality_list>

<causality_list> ::= <causality>
  <causality_list>

<causality> ::= <ipc_name> bi_temporal_operator <ipc_name>
  <bi_temporal_operator> ::= PRECEDES | FOLLOWS
    ACCESSES | MODIFIES
    NEEDS_VALUE_OF
    SENDS_INFORMATION_TO
    CREATES
    ACTIVATES
...
```

### Figure 5 Syntax of the causality part

Suppose that buffer_read IPC consists of two sub-IPCs named disk_interface, page_reader. And disk_interface sub-IPC performs interactions between buffer and disk; page_reader moves one page from disk to buffer. Then the causality in these two sub-IPCs can be represented as:
The behavior part explains how the function stored can be accomplished. Note that the objectives of IPCs is not to produce executable program codes, but to supply reusable functional information about DBMSs and other information useful for behavior simulation, dynamic analysis and semantic checking. Thus the behavior part needs to store abstracted information about a function, instead of program codes.

Design principles of IPCs include behavior analysis and measurement of DBMSs under design, as introduced in section 3. As the fundamental mechanism of behavior simulation is finite state machine, we model the behavior part to be able to produce state changes. Thus the behavior part consists of three sub-parts: PRE_CONDITION, POST_CONDITION and a formal description of the function of theIPC. The PRE_CONDITION and POST_CONDITION sub-parts are adapted from the well-known approach to verifying the consistency of a function[4, 10]. When generating a behavior of an IPC according to the input stimuli, it is necessary to check whether the input is valid, if the enabling condition of the IPC is satisfied and if the resulting state is correct. The PRE_CONDITION can check the validity of the input condition and the enabling condition. The POST_CONDITION can test the correctness of the output state. During execution of the IPC, its execution history must be generated and recorded. As the behavior specification language is not yet completed, a semi-formal syntax of the behavior part and an example of buffer_read is shown in Figure 6 and 7, respectively.

In particular, the PRE_CONDITION and POST_CONDITION parts are strengthened to check the valid state of the IPCs, besides checking ordinary boolean expressions. The checkable states includes TRANSACTION_BEGIN, TRANSACTION_ABORT, DATABASE_OPENED, DATABASE_CLOSED and so on. Other meaningful states can be added to the list of IPCs' states as the development of DBMSs is performed further.

It is possible to check the validity of the states of IPCs, because the meaning of these states comes from the domain knowledge of DBMSs. Behavior analysis and measurement of DBMSs can be done by simulating and analyzing the behavior of IPCs with respect to the intended behavior. The two behaviors are represented as a sequence of these state changes.

\[
\text{PRE_CONDITION} <\text{condition}> \\
\text{POST_CONDITION} <\text{condition}> \\
\text{FUNCTION} \{ \text{a formal description of the function of the ipc} \}
\]

\[
<\text{behavior_part}>::= \\
\text{PRE_CONDITION} <\text{condition}> \\
\text{POST_CONDITION} <\text{condition}> \\
\text{FUNCTION} \{ \text{a formal description of the function of the ipc} \}
\]

\[
<\text{condition}>::= <\text{boolean_expressions}> \\
\text{PRE_CONDITION} <\text{condition}> \text{STATE-IN} <\text{states}> \text{NONE} \\
<\text{boolean_expressions}>::=(\text{a set of ordinary boolean expressions}) \\
<\text{temporal_operator}>::= \text{ALWAYS} \\
\text{AFTER} \text{BEFORE} \text{WHENEVER} \\
<\text{states}>::=(\text{a set of user-defined states meaningful to the ipc. Examples are SYSTEM_RUNNING, SYSTEM_DOWN, TRANSACTION_BEGIN, TRANSACTION_ABORT, TABLE_INITIALIZED, TABLE_CLEARED, DATABASE_OPENED, DATABASE_CLOSED, FILE CREATED, FILE OPENED, FILE_CLOSED, etc.})
\]

<Figure 6 Formal syntax of the behavior part>

\[
\text{IPC buffer_read}
\]

\[
\text{TELEOLOGY} \text{read a page from disk to buffer.}
\]

\[
\text{STRUCTURE}
\]

\[
\text{IMPORT FUNCTION}
\]

\[
\text{IMPORT DATA STRUCTURE} \text{disk, bufferpool, page}
\]

\[
\text{EXPORT FUNCTION buffer_read}
\]

\[
\text{EXPORT DATA STRUCTURE} \text{NONE}
\]

\[
\text{CONSIST_OF disk_interface, page_reader}
\]

\[
\text{CAUSALIT}
\]

\[
\text{disk_interface SENDS_INFORMATION_TO page_reader}
\]

\[
\text{disk_interface ACTIVATES page_reader}
\]

\[
\text{BEHAVIOR}
\]

\[
\text{PRE_CONDITION}
\]

\[
\text{disk'S STATE-IN DATABASE_OPENED & FILE_OPENED}
\]

\[
\text{POST_CONDITION bufferpool'S STATE-IN PAGE_VALID}
\]

\[
\text{FUNCTION}
\]

\[
\text{check(page number is valid)}
\]

\[
\text{if not (page in bufferpool) disk_interface(move, page_number)}
\]

\[
\text{fill(moved_page, bufferpool)}
\]

\[
\text{END_OF_IPC buffer_read}
\]

<Figure 7 Example IPC: buffer_read>

3. Qualitative Behavior Modeling of IPCs

Qualitative modeling of physical systems can be characterized as representation of their behavior through differential equations for describing the physical structure of a physical system. Thus, qualitative modeling of the systems can be performed conveniently by transforming the differential equations into qualititative constraints. As a result, it is essential to provide abstraction constructs for representing the structure of a software system, such as the differential equations, before its qualitative modeling.

This section investigates the abstraction methods and various definitions for qualitative modeling of software...
systems. The nature of physical and software systems seems very different, but they are both a realization of mechanism in the real world. The mechanisms are such things as laws describing physical processes or local components that can be described as operating according to such laws.

Important constructs for qualitative modeling of physical and electrical systems considered in this paper are lumped models of primitive components, parameters (variables) of systems, structural equations, causality and behavior definitions. What is qualitatively modeled in qualitative physics is domain knowledge about physics. We choose database management systems, as a domain knowledge for software systems, whose reusable software components are proposed in section 2.

3.1 Parameters of information processing components

The parameters of qualitative modeling and simulation are structural ones that must be able to characterize the states, or the interactions of systems. For instance, the parameters for electrical systems is represented as voltage and current quantities. Thus, domain-specific knowledge for analyzing electrical networks can be represented in a unified way. The domain knowledge for the analysis is divided into two parts: network laws and device models. The network laws describe how current and voltage quantities of connected components interact, while the device models describe the behavior of a component via the voltage and currents associated with its terminals. Current is measured going into a component's terminals and voltage is measured between network nodes.

If there are notions such as voltage and current, it is possible to model the parameters in a unified way. Currently, the state of a software system is represented by the states of its variable. Thus, the parameters of information processing components are determined to be represented by a set of the states of their sub-IPC components, which includes important data structures and other information processing components.

The example parameters of buffer_read IPC shown in Figure 7 are {disk, file, bufferpool, page, disk_interface, page_reader}. Other set of parameters of buffer_read are also possible according to the degree of detailed design. That is, the dynamics of buffer_read IPC is represented by the behaviors of the six IPCs.

Each parameter must be represented qualitatively in terms of the reasonable function \( f: [a, b] \rightarrow \mathbb{R}^n \). As the states of software systems is discrete, rather than continuous, we assume that the parameters have such reasonable functions where range is discrete, which are named discrete reasonable functions. As only interested in the discrete state transitions, such notions as 'continuity' and 'continuously differentiable' are meaningless in the discrete reasonable functions. Thus, the discrete reasonable functions is defined as definition 1.

**Definition 1.** For \( [a, b] \subseteq \mathbb{R} \), define \( f: [a, b] \rightarrow \mathbb{R}^n \) to be a reasonable function if

1. \( f \) is continuous on \([a, b]\),
2. \( f \) has only finitely many critical points in any bounded interval.
3. \( \mathbb{R}^n \) is a set of meaningful states.

3.2 Lumped models of IPCs

Lumped models play the role of abstracting the mechanisms in a modular way. Electrical systems employ primitive components such as resistors, condensers and coils to describe their operations. A good explanation about lumped models is presented in [11]. The explanation is as follows.

"An integrated circuit is implemented as a semiconductor wafer with different ions diffused or implanted into its surface. The equations necessary to describe the complex circuit at the device physics level are not easily solvable and, more importantly, would not provide the designer with much insight into the overall circuit behavior."

In circuit theory, the complexity of these equations is reduced by modeling a region of space with a uniform electrical behavior as a single lumped element (e.g., a resistor, capacitor or mosfet). Each element has a set of terminals which allows it to interact with other elements. The electrical behavior of an element is described by a set of constitutive relations between state variables associated with the element's terminals.

Note that information processing components have been proposed as primitive constructs of database management systems. They can be defined as the results of functional modularization of database management systems. This definition means that the region of uniform operational behavior can be identified and represented into information processing components. Thus, lumped models for information processing components must model the uniform operational behavior in an abstracted form.

As the functions stored in information processing components correspond to mechanisms, the lumped models are determined to represent the laws related to the functions. Instead of finding out the laws whose shape is similar to that of physics or electronics, it seems reasonable to render new forms of the laws in software systems. As a basis of constructing lumped models of information processing components, it is necessary to identify the parameters characterizing the behavior of the lumped models, their landmark values and their qualitative state. The three items are argued to be sufficient to perform qualitative modeling and simulation of database management systems.

3.3 Identification of meaningful landmark values

All of the states of information processing components are candidates of the landmark values of the lumped models. Their full set of possible landmark values can be represented as a Cartesian products of their states and negated states of imported and exported functions and data structures. Among them, the meaningful landmark values are those that transform the state of an information processing component into a new state. For example, buffer_read ICP has three input data structures including disk, bufferpool and page. Thus the possible landmark values of the buffer_read ICP can be described in 4-tuple <disk, file, page, bufferpool>

It is assumed that the possible states of bufferpool are (BUFFERPOOL_EMPTY, BUFFERPOOL_FULL); the possible states of disk are (DATABASE_CLOSED, DATABASE_OPENED); the possible states of file are (FILE_CLOSED, FILE_OPENED); the possible states of page is (BUFFER_ALLOCATED, PAGE_VALID). Thus
the number of possible landmark values of the buffer_read IPC is 25. Some of the possible states of buffer_read IPC is shown in Figure 8.

The identification of meaningful landmark values is assisted by its pre-conditions and post-conditions, because of temporal qualitative modeling of information processing components. The idea is that all of state combinations not satisfying pre-condition cannot affect the starting of the performance of IPCs. And all of state combinations not satisfying post-condition cannot affect the ending of the performance of IPCs.

The set of possible landmark values =
<DATABASE CLOSED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<NOT DATABASE CLOSED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<DATABASE OPENED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<NOT DATABASE CLOSED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<DATABASE CLOSED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<NOT DATABASE CLOSED, FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<DATABASE CLOSED, NOT FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<NOT DATABASE CLOSED, NOT FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<DATABASE_OPENED, NOT FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
<NOT DATABASE_OPENED, NOT FILE CLOSED, BUFFER_ALLOCATED, BUFFERPOOL_EMPTY>,
...

As shown in Figure 8, its pre-conditions are "disk's STATE_IN DATABASE_OPENED", "file's STATE_IN FILE_OPENED", and "page's STATE_IN BUFFER_ALLOCATED." And its post-conditions are "disk's STATE_IN FILE_CLOSED" and "bufferpool's STATE_IN PAGE_VALID." That is, it is feasible to classify the possible landmark values according to the status of IPCs, as shown in Figure 9. Then the number of meaningful landmark values of buffer_read IPC can be reduced to six types. The set of meaningful landmark values is shown in Figure 10. The heuristic for defining meaningful landmark values of information processing components is as Figure 11.

3.4 Qualitative states of IPCs

The qualitative state in the qualitative simulation are a pair <qval, qdir>, where qval is landmark value and qdir is the direction of changes. As the semantics of qdir in physical and software systems are different, the qualitative states is defined as definition 2.

In the temporal qualitative modeling of software systems, it is more important to represent the state changes of a module, together with the information about the moment when the module is called. Thus, the semantics of qdir is changed to {before, during, after} of the execution time interval of information processing components. The semantics of landmark values is changed to represented the changes of states of information processing components.

Definition 2 Let l_1, \ldots, l_k be the meaningful landmark values of an parameter of an information processing component whose discrete reasonable function is f: [a, b] → R^*.

1. Define the components of possible landmark values. The name of IPCs appearing in the pre-condition and post-condition become the components.
2. Identify the set of states each component can have.
3. The set of possible landmark values is the Cartesian products of the states of the components.
4. Among the set of possible landmark values, divide the set of landmark values according to the landmark value satisfying pre- and post-conditions. The other possible landmark values are divided again into the following groups.
   1) Possible landmark values satisfying only the set of the components in pre-condition
   2) Possible landmark values satisfying only the set of the components in post-condition
   3) Possible landmark values satisfying any of the components before pre-condition
   4) Possible landmark values satisfying any of the components after post-condition
   5) Possible landmark values satisfying any of the components after post-condition
   6) Possible landmark values satisfying none of the components.

\[ Q_{\text{s}}(F, t) = Q_{\text{s}}(<f_1, \ldots, f_m>, t) \]
QS(F, t_i, t_{i+1}) = \{QS(<f_1, \ldots, f_n, t_i, t_{i+1})\}.

The qualitative behavior of F is the sequence of qualitative states of F:

QS(F, t_0), QS(F, t_0, t_1), QS(F, t_1), \ldots, QS(F, t_n)

For example, the buffer_read IPC has six parameters such as \{disk, file, bufferpool, page\}. Then the system F of buffer_read IPC is \{f_{disk}, f_{file}, f_{bufferpool}, f_{page}\}. The reason of integrating the qualitative states of individual discrete meaningful functions into a tuple is that no state change can be occurred unless all of parameters meet the pre- or post-conditions.

### 3.5 Qualitative constraints of information processing component

The purpose of qualitative constraints is modeling of the structure of a system. The structure of database management systems can be modeled as a collection of information processing components interconnected one another. Thus, there are two types of qualitative constraints: inter-components constraints and intra-components constraints. The interconnections among information processing components are modeled by inter-components constraints, while intra-component constraints describe the organization of sub-IPCs in an information processing components. A constraints can be modeled as inter- or intra-components constraints depending on the viewpoints of analyzers.

When the viewpoint of analyzer is focused on the internal behavior of one information processing component, the interior structure of the IPC is modeled as internal constraints. As there is no interconnections from/to its outside environment, the internal constraints can describe the structure only in a general way. The internal qualitative constraints are equivalent to device models of electrical components such as resistor and condenser.

In case of qualitative modeling based on differential equations, the qualitative constraints such as ADD, MULTI, MINUS and DERIV are sufficient to simulate the behavior of the qualitatively modeled systems. But, for the qualitative modeling based on finite state machines, another qualitative constraint for representing the state transitions between landmark values. We propose a new qualitative constraint called CHANGES. The formal definition of CHANGES is as follows:

**Definition 3** CHANGES(f, g) is a two-place predicate on discrete reasonable functions f, g: [a, b] → R* which holds iff f(t) changes landmark of g(i) for any t ∈ [a, b].

### 4. Conclusion

SQM is a knowledge-based software metrics management system that makes use of application domain in order to establish consistent software measurement environment. In [7], IPCs was proposed as software components for the SQM. In this paper, design principles and a formal structure, which consists of name, teleology, structure, behavior and causality parts, of IPCs have been introduced. The proposed generic structure has been refined and formally represented in BNF syntax.

As one of the application area of IPCs, we have proposed their modular, qualitative behavior generation mechanism. The proposed mechanism is based on qualitative modeling and simulation, thus produces a sequence of temporal state transitions of modeled IPCs. The proposed modeling mechanism aims to facilitate the scientific measurement of DBMSs. Further works include implementation and evaluation of the proposed qualitative behavior generation mechanism.

### References