A Constraint Programming Approach for Workflow Applications

Shigeo KANEDA* and Mitsuhiro TSUJI**

*Graduate School of Policy and Management, Doshisha University, Kyoto, 602-8580 JAPAN
**Faculty of Informatics, Kansai University, Takatsuki-shi, 569-1095 JAPAN

ABSTRACT: This paper proposes a constraint programming approach for workflow applications in which a slip is regarded as a tuple of the RDB model. A workflow is considered to be a series of database updating and constraint maintenance processes in this case. The proposed method utilizes pre-conditions and post-conditions in a constraint programming style for an agent that interprets and executes these constraints. The method has been experimentally applied to a real application and the following advantages have been confirmed: (1) No programming for opposite directional flow to correct the value of an erroneous item when an error is detected in the workflow downstream stage, and (2) Small semantic gap between the domain knowledge and the declarative programming description.

1. Introduction

This paper focuses on workflow applications of a special type in which a slip flows between users and blanks of the slip are additionally filled in by the users. Each user's action is, obviously, a database updating process in the workflow, if the slip is assumed to be a “tuple” of the Relational Database Model (RDB)[9].

Workflow applications (e.g. database updating) are usually described with a procedural programming language, such as a Script, Table-driven, or the C language. This procedural approach has some disadvantages. For example, (i) Additional routines are necessary for handling an erroneous datum input at an upstream stage and detected by a downstream user, and (ii) Confirmation on the agreement between the program description and background business rules is difficult.

To solve these problems, this paper proposes a constraint programming approach [3,4,5,6] for workflow applications. More specially, the proposed approach utilizes preconditions (data patterns to trigger a data updating process) and post-conditions (resultant data patterns generated by the data updating process) in a constraint programming style. In this approach, a constraint satisfaction agent interprets and executes these programs. As a result, the workflow is implemented in a simple and declarative style.

The proposed method has the following advantages.
(1) The size of the program is small and there is a narrow semantic gap between the program description and domain business rules.
(2) No reverse directional routine is necessary for correcting an erroneous user input.

Hereafter, Chapter 2 shows problems of the conventional procedural workflow implementation. A workflow is defined as a consistency maintenance process in Chapter 3. A new constraint programming approach is proposed in Chapter 4. Chapter 5 states the conclusion.
2. Problems of The Conventional Approach

Figure 1 shows an example of a workflow. A "purchase request slip" is first created by an employee. The items of the slip to be filling-in are the name of the employee, the name of the article to purchase, and the price. Next, this slip is sent to the section chief of the employee. If the section chief agrees with the information of this slip, the slip is sent to the head of the accounting section where the price is less than 1,000,000 yen, or sent to a director when the price exceeds 1,000,000 yen. In the latter case, the slip is sent to the head of the accounting section after approval by the director.

Each unit of processing such as the employee, the section chief, or the director in Fig.1 is called a "process" in this paper. Each process has one user. The slip flows usually in one direction as shown in Fig.1. A procedural languages appears to be suitable for this one directional application. Therefore, a procedural language, such as a Script, Table-driven, or the C language, is widely employed.

However, from the author's experience in system development, a procedural language may cause some of the following problems.

Additional Routine Requirement for Correcting An Erroneous Datum
The workflow in Fig.1 is not one-directional when a user corrects an erroneous datum. For instance, let's assume that a slip, having 990,000 yen started as the price, is sent to the accounting section. The head of the accounting section, upon receiving the slip, enters into negotiations with a seller. However, it is then found that the correct price is 1,020,000 yen. In this case, the head of the accounting section cannot sign a contract with the seller. This is because the business rules state need the director's agreement for the slip.

In this case, an additional routine is required to implement a reverse directional flow. The slip needs to flow from the accounting section to the director. Even though this is a rare case, the workflow system must be able to handle this type of opposite directional flow. This requires additional design and maintenance work for the application.

Low Readability of Program
Another problem with a procedural language approach is low "readability". For instance, the "section chief" process of Fig. 1 can be programmed as follows in a procedural language.

(Step1) The process queries the section chief's decision.
(Step2) IF (the acquisition result from the user is "accept"),
(Step3) THEN IF (the price is no less than 1,000,000 yen),
(Step4) THEN The slip is transmitted to a Director,
(Step5) ELSE The slip is transmitted to the head of accounting section.
(Step6) ELSE The processing is completed.

A major problem of this description is the medley of different types of knowledge, such as the data input (Step 1), branch (Steps 2,3), and action (Steps 4,5,6). Because of this, it is very difficult to confirm that the output of the program agrees with the background business rules.
3. Workflow as Consistency Maintenance

In this chapter, a workflow is modeled as a database updating process, and it is clarified that database updating is a type of consistency maintenance.

3.1 Workflow as Database Updating

A slip, i.e., an output of the "employee" process of Fig. 1, has the following item names and values, for instance.

```plaintext
(Employee-Name     SHIGEO-KANEDA),
(Article-Name       5GB-HARDDISK),
(Price               1,200,000),
(Section-Chief       (unbind)),
(Director            (unbind)),
(Accounting-Head    (unbind))).
```

"Employee-Name" is an item name and "SHIGEO-KANEDA" is its value. "(unbind)" shows that the item has not been input, that is, an unbound state.

In the next step, the section chief adds his/her decision, "accept" or "reject", to this slip. Evidently, this is a data updating process, if the slip is assumed to be a tuple of the RDB model. The updated data is as follows.

```plaintext
(Employee-Name      SHIGEO-KANEDA),
(Article-Name         5GB-HARDDISK),
(Price                1,200,000),
(Section-Chief         accept),   ----------   modified
(Director             (unbind)),
(Accounting-Head    (unbind))).
```

Finally, the director adds a new item value, "accept" or "reject", for the "Director" item. As shown here, a workflow can be regarded as a continuous updating process of a database tuple.

3.2 Workflow as Consistency Maintenance

Essentially, the updating of a tuple (a slip in a workflow) is a data state transition from one permitted state to another permitted state. To clarify this viewpoint, a slip is assumed to be memorized as a tuple of a single relation of the RDB model. This relation is called a "Universal Relation (UR)", hereafter. It is assumed that each Di (i=0,1,2,,p-2,p-1) is the domain of an attribute. The UR is a subset of the Cartesian Product D0 x D1 x D2,,x Dp-1, and represents the real world in the application domain.

On the other hand, the business rules never permit the memorization of "ALL" data states of the Cartesian Product. For example, let's assume that an employee's name appears twice in one slip. Let Dorig and Dcopy be the domain of the original name item and the copied name item. In this case, obviously, an extremely limited subset of the Cartesian Product Dorig x Dcopy can be memorized in UR.

Let's consider all of the data states permitted from the domain business knowledge and call the set of all permitted states the "Universal Constraint (UC)". This UC includes a special data pattern for the case that no tuple can be found by a query using a key attribute value in UR. Note that creating, updating, or deleting of a tuple in UR is a transition from one data state to another data state in UC.

The following results are obtained from the above discussion.
Analysis-1:
The output data state of each process in a workflow should be a permitted data states under the business rules. This means that the output data state of each process in a workflow is one of the data states in UC. And, the output data state of a process is usually equal to a data state for starting another process. Accordingly, a workflow is finally a "Tour" in UC. This tour is often bidirectional for an input error correction as mentioned in Section 2.

Analysis-2:
Here, let n be the size of UC. The upper limit for the number of data transitions among the data states of UC is n(n-1). This means that the amount of descriptions with a procedural approach is of the order of the second power of n. On the other hand, if the program can be written at the UC level, the amount of description is of the linear order of n.

Our target in this paper is to implement a workflow application system with a declarative description, being equal to this Universal Constraint UC. Of course, the enumeration of all actual concrete values of UC is not a good policy. The size of UC is too large. Therefore, it is necessary to suppress the size of UC to be of the order of the number of processes appearing in the workflow. Accordingly, data patterns in UC are described using "variables" as will be mentioned later.

Figure 2. Outline of the Proposed Approach

4. A Constraint Programming Approach for Workflow Applications

4.1 Outline of the Proposed Approach

From the above discussion, a new approach is demonstrated to implement a workflow in a constraint programming style [1,2,10]. A slip is a tuple on a single relation (UR) in the proposed approach as shown in Fig. 2. In addition, each process has only two declarative programming descriptions, "Precondition" and "Post-condition", both being a subset of the data patterns UC. A constraint satisfaction agent converts an input data matching with the precondition into a new data matching with the post-condition.

4.2 Constraint Syntax for Permitted Data Patterns

The slip for input of the section chief process in Fig. 1 is equal to the output data state of
the employee process. The "Employee-Name" item, "Article-Name" item, and "Price" item can be filled in with any value. Then, these values are expressed as independent variables [7]. Obviously, these variables should be bound.

The input data patterns of the section chief process can be expressed as follows,

\[
((\text{Employee-Name} \quad \text{*Employee-Name}) \text{ where } *\text{Employee-Name} \text{ should be bound,} \\
(\text{Article-Name} \quad *\text{Article-Name}) \text{ where } *\text{Article-Name} \text{ should be bound,} \\
(\text{Price} \quad *\text{Price}) \text{ where } *\text{Price} \text{ should be bound,} \\
(\text{Section-Chief} \quad \text{unbind}), \\
(\text{Director} \quad \text{unbind}), \\
(\text{Accounting-Head} \quad \text{unbind}))).
\]

Each "symbol" (for instance, "Employee-Name") indicates a data item name and each "*symbol" (for instance, "*Employee-Name") indicates a variable. In the same way, the permitted input/output data states of the other processes can be expressed.

4.3 Complete Program for a Process

The complete programming description for a process is clarified in this subsection. First, each process has a condition to start the process. The condition consists of one or more disjoint data patterns. The data patterns are called the "Precondition". Moreover, each process has a completion condition. This condition is also a set of disjoint data patterns. These data patterns are called the "Post-Condition". For instance, the pre/post-conditions of the section chief in Fig. 1 are as follows. The variable name has validity only in one data pattern. Therefore, even if the same variable name is assigned, its value may not be equal to the each other.

**Preconditions: (One data pattern in this case)**

\[
((\text{Employee-Name} \quad \text{*Employee-Name}) \text{ where } *\text{Employee-Name} \text{ should be bound,} \\
(\text{Article-Name} \quad *\text{Article-Name}) \text{ where } *\text{Article-Name} \text{ should be bound,} \\
(\text{Price} \quad *\text{Price}) \text{ where } *\text{Price} \text{ should be bound,} \\
(\text{Section-Chief} \quad \text{unbind}), \\
(\text{Director} \quad \text{unbind}), \\
(\text{Accounting-Head} \quad \text{unbind})).
\]

**Post-condition: (Two data patterns in this case)**

\[
((\text{Employee-Name} \quad \text{*Employee-Name}) \text{ where } *\text{Employee-Name} \text{ should be bound,} \\
(\text{Article-Name} \quad *\text{Article-Name}) \text{ where } *\text{Article-Name} \text{ should be bound,} \\
(\text{Price} \quad *\text{Price}) \text{ where } *\text{Price} \text{ should be bound,} \\
(\text{Section-Chief} \quad \text{accept}), \\
(\text{Director} \quad \text{unbind}), \\
(\text{Accounting-Head} \quad \text{unbind})).
\]

\[
((\text{Employee-Name} \quad \text{*Employee-Name}) \text{ where } *\text{Employee-Name} \text{ should be bound,} \\
(\text{Article-Name} \quad *\text{Article-Name}) \text{ where } *\text{Article-Name} \text{ should be bound,} \\
(\text{Price} \quad *\text{Price}) \text{ where } *\text{Price} \text{ should be bound,} \\
(\text{Section-Chief} \quad \text{reject}), \\
(\text{Director} \quad \text{unbind}), \\
(\text{Accounting-Head} \quad \text{unbind})).
\]

The first pattern in the above post-condition is a condition of termination of the section chief process. The flow of the second one depends upon the price. That is, the second pattern becomes the precondition of the director process when the price exceeds 1,000,000 yen. Otherwise, it is the preconditions of the head of the accounting section when the price is less than 1,000,000 yen. A correspondence to Fig. 1 can be seen where the flow diverges from the section chief process. As shown here, the post-condition of a certain process diverges into plural precondition of other processes when there is flow divergence.
4.4 Constraint Satisfaction Agent

A database updating agent, that is, a constraint satisfaction agent, executes the data transition in each process of workflow applications. Only the precondition and post-condition are described for constraint programming. Here, it is assumed that the precondition and post-condition are disjoint. This is appropriate if the process is to execute some data modification. When a data state satisfies the precondition in a process, the same data never matches the post-condition of the same process. It is the responsibility of the database updating agent to convert this contradicted state into a non-contradicted state. Note that all of the domain knowledge is concentrated into the post-condition. Therefore, the database updating agent should not add more restrictions to the precondition data. The agent should accomplish the following simple functions.

Functions of Database Updating Agent

Function-1: The agent is invoked if the tuple data matches its precondition. The matching mechanism is "Unification". When the data matches one of the post-conditions, however, the agent stops the execution. The responsibility of the agent is to specify only one data pattern in the post-condition.

Function-2: If the agent detects a contradiction, the agent presents all of the data to the user to accomplish function-1 above. The user inputs or modifies the data and converts the data state into one of the post-condition.

Function-3: When the user input value satisfies only one pattern in the post-condition, the other items are rewritten into the value of the matched post-condition. To accomplish this function, each data item has a "strength" value. Old data satisfying the pre-condition has a "weak" strength value and data newly input has a "strong" strength value. When only one data pattern in the post-condition can be unified with a "strong" item value, the data items with a weak strength value are converted to the value of the matched data pattern in the post-condition.

Basic Operation

The above agent movement is shown by an example. First, it is assumed that the following data exists on the tuple of the section chief process in the workflow of Fig.1.

((Employee-Name SHIGEO-KANEDA),
(Article-Name 5GB-HARDDISK),
(Price 1,200,000),
(Section-Chief (unbind)),
(Director (unbind)),
(Accounting-Head (unbind))).

This data can be unified with the pre-condition of the section chief process. As a result, the section chief agent is invoked.

When an agent is invoked, the data state does not agree with either of the post-conditions. The agent has the obligation to make the data state match either of the post-conditions. In this case, it is not possible to judge which of accept or reject is selected for the section chief decision by the agent alone. Therefore, the agent presents all of the data to the user. If the section chief inputs "accept" as his/her decision, for example, because only one data pattern in the post-condition was been approved, the processing of the agent is completed.

Automatic Data Transfer

If "data item strength" is introduced into each data item, the automatic correction of the data is also possible. For instance, let's consider the following original data state. It is also
assumed that the "Message" item is "Your section-chief gives his/her approval", if the "Section-Chief" item is "Accept" in this example.

Original Data:

((Employee-Name   (strength weak)   SHIGEO-KANEDA).
(Article-Name      (strength weak)   5GB-HARDDISK).
(Price             (strength weak)   1,200,000).
(Section-Chief      (strength weak)   (unbind)).
(Message          (strength weak)   No-Message).
(Director          (strength weak)   (unbind)).
(Accounting-Head   (strength weak)   (unbind))).

Post-Condition: (only one data pattern is displayed.)

((Employee-Name     *Employee-Name) where *Employee-Name should be bound,
(Article-Name      *Article-Name) where *Article-Name should be bound,
(Price             *Price) where *Price should be bound,
(Section-Chief     accept),
(Message  Your section-chief gives his/her approval),
(Director          (unbind)),
(Accounting-Head   (unbind))).

The above original data cannot unify with the above post-condition.
In this case, the agent presents all of the original data to the user. The user selects "accept" as the "section-chief" item.

((Employee-Name      (strength weak)   SHIGEO-KANEDA),
(Article-Name         (strength weak)   5GB-HARDDISK),
(Price                (strength weak)   1,200,000),
(Section-Chief         (strength strong)  accept),
(Message              Your section-chief gives his/her approval),
(Director             (unbind)),
(Accounting-Head      (unbind))).

This data pattern cannot unify with the above post-condition, too. However, the strong data item "accept" requires that the post-condition be satisfied. Therefore, the weak "message" item is rewritten into the value for the post-condition, "Your section-chief gives his/her approval".

A notable feature of the proposed method is its declarative style. No other additional programming except data structure definition is required to implement a workflows. This table is equivalently equal to the universal constraint UC. However, each data pattern appears twice because this table has pre/ post-conditions.
Let n be the number of data patterns in UC. The amount of descriptions in the proposed method is obviously 2n. On the other hand, a procedural approach requires n(n-1) descriptions. Therefore, if a workflow has only one directional flow, the conventional procedural approach is superior to the proposed method. On the other hand, if the workflow has many opposite directional paths, the proposed method is superior.

5. Conclusion

A new constraint programming approach was demonstrated. A workflow application is regarded as a series of database updating and constraint maintenance processes. As a result, data patterns satisfying the business rules are described as a constraint program.

The proposed technique was experimentally applied to a workflow. The following advantages were confirmed. (1) There is no opposite directional flow to correct the value of an erroneous item when an error is detected at a downstream stage, and (2) a small semantic gap exists between the domain knowledge and the declarative programming description.
Acknowledgement

The authors thank you very much Mrs. Megumi ISHII for her contribution and discussion. Also, the authors thank Mr. Teiji EMORI and Mr. Satoshi WATANABE, Nippon Telegraph and Telephone Advanced Technology Corporation for their support and guidance.

References