representation which is easy to read and maintain. Second, it avoids the overhead of directly interpreting the input of the programmer by translating that input to a PTN. Last, the PTN hides some of the proprietary information needed to reconstruct the business process. With a simple and powerful mechanism on the execution level it is possible to migrate the execution of workflow processes as well as to support every "structure" needed on the definition level. Since there is currently no established standard for the low-level description of the control flow of workflow processes, we think that PTNs have a good chance to fill this gap.

We are currently working on the depiction of further high-level constructs as e.g. the *par-foreach* statement on a PTN. Furthermore we are investigating the structure of so-called "groupware working modes" and their depictability on both high-level programming constructs and PTNs.

Another working line in the development of APRICOTS is the connection of the ConTract Manager with a transactional environment like Encina from Transarc. This work aims at the definition of a "handbook" to enhance transaction monitors by mechanisms to run workflows directly concerning advanced transaction models like e.g. the ConTract model. Likewise we are looking at the upcoming standard of the Workflow Management Coalition to extend their approach with transactional semantics.

Since the external manipulation commands of a running process are currently limited to *start*, *stop*, *suspend* and *resume,* we are also working on the development of concepts to handle arbitrary (external) events. That means we want to integrate mechanisms in our engine to catch events from running ConTracts which will cause activities in other running ConTracts as e.g. the negotiation about resources. Furthermore the handling of time and events caused by a clock is another important point of our future work.

Last but not least we will have a look at the management of changes in the process definition while instances of this process are running. That means concepts will be developed on how and when a process instance can be changed. This work will lead us from the current view of statically predefined processes to the dynamic approach which allows the evolution of a process a run time like e.g. in INCAS [1].

References

[1] D. Barbara, S. Mehrotra, M. Rusinkiewicz: INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments, Technical Report, University of Houston (TX), 1994

[2] Wilfried Brauer "Petri nets: central models and their properties; advances in Petri nets", Springer Verlag (Lecture notes in computer science). 1987

[3] U. Dayal, M. Hsu, R. Ladin "Organizing Long-Running Activities with Triggers and Transactions, ACM SIGMOD Record, pp. 204-214, 1990

[4] Ahmed K. Elmagarmid ed. "Database Transaction Models for Advanced Applications", Morgan Kaufmann Publishers, Inc., 1992

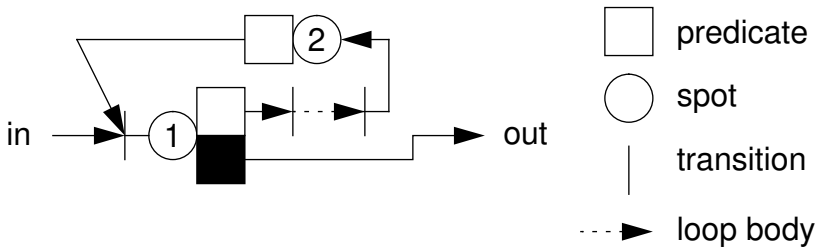
[5] Friedemann Schwenkreis "APRICOTS - A Prototype Implementation of a ConTract System: Management of the Control Flow and the Communication System" Proc. of the 12th Symposium on Reliable Distributed Systems, pp.12-21, 1993

[6] Helmut Waechter, Andreas Reuter "The ConTract Model" chapter 7 in [4] pp. 219-263

different results easily with predicate controlled connections of spots and transitions.

The second decision concerned the definition of the parts of a PTN which represent the high-level constructs. For each high-level construct, we had to make a separate design. In this extended abstract we present one example: the *for-loop*.

A for-loop is defined similar to its definition in classical programming languages. There is a variable which represents a loop-counter (currently only integer variables are supported). A predicate is needed to define the end condition of the loop. Last but not least the “increment” of the loop has to be defined, i.e. in the current state of the programming environment a number has to be defined which is added to the loop-counter every time the loop is executed. The following figure represents the corresponding part of a PTN.



In this figure, the spot marked with 1 is only needed to pass a token to the two predicates and does not represent any action. The spot marked with 2 stands for the incrementation of the loop counter. It has turned out that for almost every high-level construct such “send-token” spots like spot 1 are needed which are called “pseudo-steps”. Likewise the predicate associated with spot 2 is always true and only needed to fulfill the requirements of PTNs. The predicate represented by the black square after spot 1 is the negation of the other predicate of spot1 (end condition) which is again needed due to the requirements of PTNs.

As a matter of fact, the PTN generated from a high-level representation of a control flow creates redundant information. Since the visualization during the execution of processes should be done on the basis of the high-level representation, this can be a problem because some elements of the PTN are should not be seen on the user’s level. These problems are solved in APRICOTS by assigning types to spots and predicates to be able to differentiate in elements which are relevant for the visualization and which are not.

Due to the fact that APRICOTS supports the user with a transactional environment - on the programming level as well as on the execution level - we decided to use a relational DBMS as a stable storage of the whole system. That means that templates and instances of ConTracts are stored in a RDBMS. The data needed (loop counters, number of expected tokens, etc.) during run-time is read from the RDBMS and the changes to this data are recorded in the RDBMS. This approach has the advantage of high portability and, what is more important, of allowing state changes of an instance to be done under the protection of transactions as they are defined on the *Script* level.

5 Summary and future work

The distinction of two levels of representation of a workflow definition has several advantages and is requested by users. First, it provides programmers with a simple graphical

- blocks (similar to macro definitions)

All additional information needed to define the objects properly can be entered with specialized dialog boxes. For example, if the user opens the property dialog box of a *Step,* he or she can enter the special information associated with *Steps*:

- Parameters needed by the *Step* and the corresponding *Context* variables
- Invariants which are needed for a proper isolation
- The definition of the conflict resolution policy to resolve concurrency conflicts
- *Steps* which should be started if the transaction of the current *Step* is aborted or a conflict resolution is finished
- The logical name of the associated *Step-Server* which has to be called to execute the *Step.*

The main advantages of the newly developed editor are the object-orientation and the direct usage of graphical elements for programming. Through the object-orientation the programmer of a ConTract is able to use and manipulate well-known constructs of high-level programming languages instead of “drawing a net”. Furthermore the editor can easily be enhanced by new, not yet supported, constructs like e.g. the *parforeach* statement.

On the other hand the representation generated or used by the editor cannot be used directly for the execution. Since there is too much additional information integrated to support a proper visualization and manipulation, the direct interpretation of the editor output would cause a big overhead. To solve this problem, a translator has been developed to generate a PTN out of the editor’s output. Once again, this is similar to an object-oriented approach: the editor is used to define a *template* of a ConTract which is used by the translator to generate an executable *instance* of a ConTract.

From an interoperability point of view the distinction between the two forms of the definition of a ConTract has another advantage: the people who define a workflow (e.g. workflow consultants) are very interested in the “migratability” of their processes, i.e. that a defined process should be executable on several platforms with different workflow engines. On the other hand, they want to avoid that the original business process (which is modelled in the programming environment) can be reconstructed from the information used by workflow engines. By using a very low-level mechanism on the execution level, enough information from the original definition can be hidden to avoid the unauthorized reconstruction of business processes (this is similar to the differentiation between source code and binaries of usual programming).

4 From the programming to the executable representation

Several design decisions have been made to realize a proper representation of a ConTract’s control flow as a PTN. Since users want to observe the progress of their processes with the same graphical representation as they have used during the programming phase, some information from the original definition must be integrated with the predicate transition net. Furthermore, it is very helpful for the implementation of advanced features of the executing engine to have some additional information beyond the “raw” PTN.

The first decision in the design was to have *Steps* of a *Script* represented by spots in the PTN. Since we want to handle the result of a *Step*, we can express the different paths for

- Predicates (P)
- Transitions (T) with an associated attribute expected_tokens

Based on these elements, a directed graph can be built with the following constructs:

- Tuples (T, S) which express the connection of a transition T with a spot S.
- Triples (S, P, T) which express the connection of a spot S with a transition T, controlled by a predicate P.

The definition of a PTN can be animated by an execution mechanism which realizes the following rules:

- The execution of a PTN can be started by placing a token on a transition or a spot.
- If a transition T receives a token it *fires* if the number of received tokens is greater than or equal to the number of expected tokens. To *fire* means to pass a token to every spot S_i for which a tuple (T, S_i) exists.
- If a spot S receives a token every predicate P_i is evaluated for which a triple (S,P_i, T_j) exists. If the predicate P_i evaluates true a token is passed to the transition T_j of the triple.
- The execution of a PTN ends if no more recipients of tokens can be found when a transition wants to fire.

It is easy to recognize that PTNs are a very powerful mechanism to express the control flow of a process. They are very similar to the so-called Event-Condition-Action-rules [3] and have the same drawbacks on the programming level. Even so, PTNs are suitable for usage as a low-level representation of processes executed by workflow engines.

3 Programming in APRICOTS

As it has turned out, the original approach of ConTracts to define the *Script* by a (procedural) language is not useful. Since the users of a workflow system want to observe the progress of their processes, a mechanism to *visualize* the progress is needed. Furthermore, it is a requirement of users to have (almost) the same view of a *Script* in both the programming phase and the execution phase.

To fullfil these requirements a graphical programming environment for ConTracts has been developed in APRICOTS. This environment allows the programming and controlling of ConTracts on the basis of the same graphical representation of the control flow. The control flow is represented as a directed graph in which *Steps* are represented by ellipses, predicates by small circles and “flow paths” by lines. This approach follows the “drawing bubbles and arcs” principle but was enhanced by powerful editing functions.

The graphical editor for the definition of a ConTract is totally object-oriented and offers the following programming primitives (objects) to build up a process definition:

- *Steps*
- sequences
- loop blocks (for, while do, repeat until)
- branch blocks (if then else, n paths from m paths)
- parallel blocks
- transaction boarders

Submitted to the 6th Intern. Workshop on High Performance Transaction Systems (HPTS)

Extended abstract of work in progress

APRICOTS - a workflow programming environment

Friedemann Schwenkreis
Institute of Parallel and Distributed High-Performance Systems
Department of Applications of Parallel and Distributed Systems
University of Stuttgart
Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany
email: schwenk@informatik.uni-stuttgart.de

Areas of responsibility: ConTracts, APRICOTS,
PoliFlow (workflow systems for the german federal government)
Areas of research: concurrency control, workflow mechanisms

1 Introduction

The ConTract [6] model, originally introduced as an extended transaction model, covers almost every aspect of workflow systems. In particular, the prototypical implementation of a ConTract system (APRICOTS [5]) shows the applicability of the concepts in the workflow area.

Since the programming of workflow processes is one of the most critical tasks of workflow systems, the programming environment of APRICOTS will be presented. Furthermore, the representation of the low-level description, which is directly used by the ConTract-Manager to execute a workflow process, is introduced. It will be shown that a so-called “predicate-transition-net” is a powerful means to express control flows. However, because it is difficult to read and maintain, this predicate-transition-net is not used directly by the programmer. Instead, APRICOTS provides the programmer with a graphical representation of the control flow.

We will follow a bottom-up approach to describe the programming environment. In section 2 the predicate-transition-net (PTN) is introduced. Section 3 introduces the high-level programming constructs to program a ConTract. In section 4 the depiction of the high-level constructs on a PTN will be described. In the last section we will conclude with a summary and we will give an outlook on our future work.

2 The predicate-transition-net (PTN)

A predicate-transition-net or *PTN* is a mechanism to define processes or control flows very similar to *petri nets* [2]. With the combination of the functionality of petri nets and the expressiveness of predicates a very powerful means is available to express processes in the workflow area.

There are three basic elements on which PTNs are built:

- Spots (S)