Integrating organisational and transactional aspects of cooperative activities*

Frans J. Faase, Susan J. Even, Rolf A. de By, Peter Apers†

University of Twente, The Netherlands

March 10, 1997

Abstract

This paper introduces the specification language CoCoA. The features of CoCoA are designed for the specification of both organisational and transactional aspects of cooperative activities, based on the CoAct cooperative transaction model. The novelty of the language lies in its ability to deal with a broad spectrum of cooperative applications, ranging from cooperative document authoring to workflow applications.

1 Introduction

CoCoA is a specification language for cooperative activities [11]. The novelty of the language lies in its ability to deal with a broad spectrum of cooperative applications, ranging from cooperative document authoring (CDA) to workflow applications (WF). CoCoA is unique in that it deals with both organisational and transactional aspects of cooperation in a single language, but without coupling them, as in *transactional workflows* [15], which assign transactional properties to the

*This research was supported by the ESPRIT LTR project TRANSCOOP (EP8012). TRANSCOOP is funded by the Commission of the European Communities. The partners in the TRANSCOOP project are GMD (Germany), Universiteit Twente (The Netherlands), and VTT (Finland).

†{faase,seven,deby,apers}@cs.utwente.nl

organisational steps of a workflow. In CoCoA, the organisational aspects of a cooperative activity are specified by means of a procedure definition mechanism, which is based on a formal state transition model. Transactional aspects are specified by means of execution rules. Termination constraints link the state of the execution rules to the transitions in the step model. The language features in CoCoA are used to extend an existing database schema for cooperative work.

CoCoA has a rich set of primitives for specifying the organisational aspects of a cooperative activity. A procedure definition specifies which operations are enabled at each stage of an activity by means of *steps*. The CoCoA concept of a step is much broader than that found in traditional workflow systems. A single step can deal with more than one user, and each user can be involved in more than one step at the same time. Also, a step can allow a user to execute many different operations, without prescribing any fixed order. This makes CoCoA suitable for specifying free forms of cooperation, such as those found in cooperative document authoring, while still being able to specify the more restricted forms of cooperation found in workflow applications. Details of the procedure definition mechanism are given in Section 4.

CoCoA specifies dynamic consistency requirements on data operation invocations. To do this, *execution rules* are used to specify the allowed orderings of invocations by means of an extended form of regular expression with operation invocation patterns. Each execution rule applies to a parameterized subset of the invoked data operations. Details of the execution rules mechanism of the language are given in Section 5.

CoCoA extends the CoAct transaction model [11], which is based on the idea of exchanging partial results between the users involved in a cooperative activity. In addition to a centralized database, each user has a workspace, in which private copies of the data reside. Users can exchange partial results with each other, or via the central database. After the completion of the cooperative activity, which can be considered as a long-lived transaction, the result of the activity is found in the shared central database. In the CoAct model, the operations performed on the data are exchanged, instead of the data itself. CoAct uses a merge algorithm [14], which exploits the commutativity properties of database operations, to allow the users in the cooperative activity to

work in parallel.

Both the specification language CoCoA [5] and the transaction model CoAct were designed during the ESPRIT TransCoop project. CoCoA is based on the object-oriented, functional database specification language TM [2]. The semantics of the language has been defined by mappings to the language LOTOS/TM [4], which is based on the process-algebraic language LOTOS [3] and TM, both of which have well-defined semantics. A tool set for CoCoA has been implemented within the TransCoop project; it includes a simulation environment, based on the TM Abstract Machine, and a compiler to a run-time environment, which consists of a CoAct transaction manager running on top of the VODAK object-oriented database system. This environment is being studied in the context of the SEPIA cooperative document authoring system [13].

2 Related work

Several extended transaction models have been designed to deal with the long-lived transactions found in cooperative systems. Examples of these are the open-nested transaction model [16], which has been implemented in the VODAK system [7], and the ConTract model [10] of which APRICOTS [12] is a prototype implementation. Although these transaction models support long-lived transactions, they are still limited in their support for truly cooperative activities.

During the past decade, traditional groupware applications, such as process-oriented workflow systems have been extended with transactional abilities. Examples of these are the TriGS$_{flow}$ workflow system [8], which is based on an active database system using a multi-parent nested transaction model, and the Exotica system [1], which shows how advanced transaction models, such as Linear Sagas and Flexible Transactions, can be implemented on top of existing workflow systems such as FlowMark.

There are similarities between the steps used in the procedure definition mechanism of CoCoA, and the state charts that are used in Mentor [17]. Although CoCoA defines control flow, it differs from many of the workflow systems, such as TriGS$_{flow}$, which assign transactional properties to steps.

In De Francesco [6], regular expressions are used to specify the external behaviour of objects in an object database, which could be compared to our approach if the entire database is considered as a single complex object. Nodine [9] describes *transaction groups* as a formal notation for the specification of cooperative transactions. An LR(0) grammar is used to describe a transaction group's correctness criteria in terms of valid histories. Neither of these approaches deals with organisational aspects or history merging. Furthermore, they do not consider constraints that depend on the states of the grammar rules.

3 An example CDA scenario

The example cooperative scenario that we use throughout this paper deals with an editor who, with the help of some co-authors, must write a document, which is reviewed by a referee. The document consist of a number of chapters with text, which can be spell-checked, and annotated. The example has been chosen such that it demonstrates sequencing, parallelism, choice, repeated activation and multiple parallel activation of steps. It also illustrates a termination condition guaranteeing that the final version of the document is spell-checked.

The overall organisation of the cooperative scenario consists of three steps: the *preparation step* during which the editor writes the title page and the introduction, the *writing step* during which the editor will assign tasks for writing a chapter to a group of authors, and the *review step* during which the referee reviews the document.

In Figure 1, the first part of the CoCoA specification for the example scenario is given. In it, only the *signatures* of the data operations have to be given. It is assumed that these operations are specified in a separate TM specification. The same holds for type definitions: only the names of the types need to be given. For simplicity, we consider the title page, the introduction, the conclusion, and the bibliography as chapters in our data model.[1]

Through so-called *communications*, the users of a scenario initiate state transitions in the

[1] Please note: This is a simplified example. Cooperation would obviously be increased if the operations were defined on paragraphs instead of chapters.

```
scenario write_document
data types  chapter, text, annotation

database operations
  addChapter(chapter, text)  remChapter(chapter)
  editChapter(chapter, text)  spellCheck(chapter)
  addAnnotation(chapter, annotation)  remAnnotation(chapter, annotation)

workspace types
  cda = { addChapter, editChapter, remChapter, spellCheck, addAnnotation, remAnnotation }

user types
  referee using cda,  editor using cda,  author using cda

communications
  introWritten(),
  startTask(chapter, ℙ author), completeTask(chapter),
  readyWriting(), documentOkay(), reviseDocument(), abortWriting()

data exchange operations
  Annotations(c : chapter) = select addAnnotation(c,_)
  Chapter(c : chapter) = select addChapter(c,_), editChapter(c,_), remChapter(c)
```

Figure 1: Interface specification of scenario

procedure of the scenario. They are called communications because state transitions influence what the other users in the scenario are allowed to do. In Figure 1 only the signatures of the communications are given. They are used in the procedure definition, as we illustrate in the next section.

The underlying CoAct transaction model uses history merging as the primary principle for its operation. To allow the user to easily select data operations from the history related to a particular piece of data, selection operations can be defined, which can be called by the users during an import or export operation. Two such data exchange operations are defined in Figure 1. One for the exchange of annotations, the other for the exchange of text changes. The symbol '_' is used to indicate that no restrictions are placed on the value of a parameter in that position.

4 Organisational aspects

The procedure according to which the scenario is executed is defined in Figure 2.[2] In the header of the procedure the parameters which identify the users of the scenario are given. Furthermore, the header contains the interaction points of the scenario. When the scenario is instantiated, the `start` interaction point is enabled. The scenario can terminate with two possible outcomes: `cancel` and `done`. Right at the start of the body of the procedure, the name and type of the shared workspace are given.

Inside the body of the procedure a number of steps are defined, of which some steps contain nested steps. The three top level steps are `preparation`, `writing` and `review`. Each of these steps has incoming and outgoing interaction points, which are sequenced through transitions defined at the bottom of procedure specification. A step is activated when an in-coming transition occurs, and deactivated when an outgoing transition occurs, or when its surrounding step is deactivated. When, for example, the transition specified by '**on** `preparation.done` **do** `writing.start`' occurs, then the step `preparation` is deactivated and the step `writing` is activated. This is an example of sequential steps.

Whenever a step is activated enable expressions specify which data operations, data exchange operations, and communications are enabled for the users of the scenario. Enabled operations can be invoked any number of times. Only by invoking communications (once enabled) one or more steps can be activated or deactivated. The enable construct inside the body of the step `writing` enables two event-condition-action rules, which specify how the tasks are activated by the editor by means of the communication `startTask`, and how the editor can terminate the whole step through the communication `readyWriting`.

Unlike transactional workflows, no transactional properties are assigned to steps, which means that the effects of the data operations which were enabled by a step are not affected when the step is deactivated.

[2] Please note that dots are used to indicate parts that have been omitted from the specification.

```
procedure (ref : referee, ed : editor,
          authors : ℙ author)[in start out cancel, done]
begin workspace  document : cda

  step preparation[in start  out done] ...

  step writing[in start  out done]
    parallel(ch : chapter)
      step task[in start(ℙ author)  out compl] ...
    endpar

    on start enable
       when ed issues startTask(c, a_s) iff (as subset authors) do task(c).start(a_s),
       when ed issues readyWriting() do done
    endon
  end

  step review[in start  out accept, reject, revise] ...

  on start do preparation.start           on review.accept do done
  on preparation.done do writing.start    on review.revise do writing.start
  on writing.done do review.start         on review.reject do cancel
end
```

Figure 2: Procedural specification of scenario

5 Execution rules

To ensure that data consistency is preserved in the workspaces, *execution rules* and *history rules* are used. The execution rules pose additional constraints on the order in which invoked data operations can occur in histories of each of the workspaces in isolation. These histories can both contain data operations executed by the owner of the workspace, and data operations which where imported from any of the other users. The ordering constraints are expressed by means of extended regular expressions. The elements of a regular expression are data operation patterns, which can include values and variables for the parameters. A history is correct with respect to an execution rule, if and only if it is a prefix of the histories generated by the regular expression, when ignoring all data operation invocations not matching any element in the rule. The following order rule from our example states that a chapter can only be edited, spell-checked, or removed after it has been added to the document, and that a chapter cannot be edited or spell-checked anymore when it has been removed. There is no restriction on how often a chapter may be edited and spell-checked.

data operation order

```
chapter_rule :
```

```
forall c : chapter
order addChapter(c,_) "edited";
        (editChapter(c,_) "edited" | spellCheck(c))*;
        remChapter(c)
```

The string "edited" after the addChapter(c,_) and editChapter(c,_) data operation patterns, is used to indicated that the chapter is in the edited state, directly after these data operations have been carried out. A query expression on the states of the execution rules can be attached to the transitions between steps in the procedure definition to describe break-point and termination constraints. For example, a query expression on the state of the above rule can be added to the condition of the final transition of the procedure definition to enforce that all chapters have been spell-checked. This is explained in the full paper.

6 Conclusions

In this paper we describe CoCoA, a specification language for data-intensive cooperative applications; the language is based on the CoAct cooperative transaction model. CoCoA takes an alternative approach in combining organisational aspects with transactional aspects of cooperative activities, compared to other advanced transaction models.

During the design of CoCoA, special attention has been paid in defining the semantics of the language through mappings to two well-defined formal languages: the process-algebraic language LOTOS, and the database specification language TM. Within the ESPRIT TransCoop project, a prototype implementation of a cooperation manager that implements the organisational aspects of CoCoA specifications, and of the CoAct transaction model have been made. These are being tested with a demonstrator application based on the SEPIA cooperative hypertext authoring system.

References

[1] G. Alonso, D. Agrawal, A. El-Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *Proceedings of the 12th International Conference*

on Data Engineering, pages 574–583, Washington - Brussels - Tokyo, February 1996. IEEE Computer Society.

[2] H. Balsters, A. de By R. and R. Zicari. Typed sets as a basis for object-oriented database schemas. In Oscar M. Nierstrasz, editor, *Proceedings of the Seventh European Conference on Object-Oriented Programming, LNCS #707*, pages 161–184, Kaiserslautern, Germany, 1993. Springer.

[3] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[4] S.J. Even, F.J. Faase, and R.A. de By. Language features for cooperation in an object-oriented database environment. *International Journal of Cooperative Information Systems*, 5(4):469–500, 1996.

[5] Frans F. Faase, Susan J. Even, and Rolf A. de By. An introduction to CoCoA. Technical Report INF-96-10, University of Twente, September 1996.

[6] Nicoletta De Francesco and Gigliola Vaglini. Concurrent behavior: A construct to specify the external behavior of objects in object databases. *Distributed and Parallel Databases*, 2(1):33–58, January 1994.

[7] GMD, Germany. *VODAK Manual, Release 4.0*, 1995.

[8] G. Kappel, S. Rausch-Schott, W. Retschitzegger, and S. Vieweg. TriGS: Making a passive object-oriented database system active. *Journal on Object-Oriented Programming*, (4), July 1994.

[9] H. M. Nodine, S. Ramaswamy, and S. B. Zdonik. A cooperative transaction model for design databases. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 3. Morgan Kaufmann Publishers, Inc., 1992.

[10] Andreas Reuter and Helmut Wächter. The ConTract model. *IEEE Data Engineering Bulletin*, 14(1):39–43, Mar 1991.

[11] Marek Rusinkiewicz, Wolfgang Klas, Thomas Tesch, Jürgen Wäsch, and Peter Muth. Towards a cooperative transaction model - the cooperative activity model -. In *Proceedings of the 21th VLDB Conference*, Zurich, Switzerland, Sep 1995.

[12] F. Schwenkreis. Apricots – A prototype implementation of a ConTract system – Management of the control flow and the communication system. In *Proceedings of the 12th Symposium on Reliable Distributed Systems, Princeton (NJ)*. IEEE Computer Society Press, 1993.

[13] N. Streitz, J. Haake, J. Hannemann, W. Schuler A. Lemke, H. Schuett, and M. Thuering. SEPIA: A cooperative hypermedia authoring environment. In *Proceedings of the ACM Conference on Hypertext (ECHT'92), Milano, Italy*, pages 11–22, 1992.

[14] Jürgen Wäsch and Wolfgang Klas. History merging as a mechanism for concurrency control in cooperative environments. In *Proceedings of the 6th IEEE CS Intl. Workshop on Data Engineering: Interoperability on nontraditional Database systems RIDE-NDS'96*, pages 76–85, 1996.

[15] Gerhard Weikum. Extending transaction management to capture more consistency with better performance. In *Proceedings of the 9th French Database Conference*. 1993.

[16] Gerhard Weikum and Hans-Jörg Scheck. Concepts and applications of multilevel transactions and open nested transactions. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 13. Morgan Kaufmann Publishers, Inc., 1992.

[17] Dirk Woedtke, Jeanine Weissenfels, Gerhard Weikum, and Angelika Kotz Dittrich. The mentor project: Steps towards enterprise-wide workflow management. In *In: Proc. of the 12th IEEE International Conference on Data Engineering, 1996 ICDE'96*, February 1996.