Criteria-Based Software Process Transaction Model *

SHEN Bei-jun[1,2], CHEN Cheng[2], JU De-hua[1,2]

[1](*Department of Computer Science, East China University of Science and Technology, Shanghai* 200237, *China*);

[2](*Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: beijun.shen@netease.com; dh.ju@computer.org

Received April 25, 2001; accepted July 17, 2001

Abstract: Transaction management is the key component of process-sensitive engineering environments (PSEE). In recent years, several advanced transaction models have been proposed to support long transaction. However, in view of specific characteristic in software process transaction, those models only meet its partial requirements. Moreover, no commercial or academic results of nested cooperative transaction models have reached a status stable enough for commercial implementation. It remains a real challenge to the transaction mechanism of PSEE. In this paper, a criteria-based transaction model E-Process/TM is presented, that may address the key features of software process, i.e. interactive user control, long-duration activities, iterative approach and multi-user cooperation on shared persistent data. Based on user defined correctness criteria, E-Process/TM offers inherent benefits in flexibility and openness. By now, this model has been implemented in a commercial PSEE product, and applied in practice successfully.

Key words: software process; software process transaction; extended transaction model; PSEE; correctness criteria

Nowadays it is widely accepted that quality is not only related to the product, but to the organization and to the production process. This view forms the core of Total Quality Management (TQM). Process-Sensitive engineering environments (PSEE) have been developed to meet this requirement, which supports process modeling, analysis and simulation, enactment, assessment and improvement automatically.

A web-based distributed PSEE, called “E-Process”, has been and being researched and developed by a joint team from East China University of Science and Technology, SRA Japan and ASTI Shanghai[1]. It intends to help software industry to establish a basic framework of enterprise process management and a strong software process infrastructure for continuous improvement, assisting project management, supporting team cooperation and collaboration, and following SE discipline to raise software quality.

Besides visual process modeling language, distributed architecture and process evolution mechanism, transaction management all belong to key technologies[2] for developing E-Process. Firstly this paper analyzes main features of software process transaction. Secondly, a criteria-based software process transaction model is proposed in detail, including the definition of software process transaction and eight correctness criteria. Then related work is discussed. At last it summarizes the characteristics and advantages of the model.

---

* **SHEN Bei-jun** was born in 1969. She is a Ph.D. candidate at CAS, and a lecturer in the Department of Computer Science and Technology, East China University of Science and Technology. Her research interests are software engineering, software process, software tools and enterprise network computing. **CHEN Cheng** was born in 1977. He is a Ph.D. candidate at CAS. His research interests are software engineering, software process and software architecture. **JU De-hua** was born in 1938. He is an IAB member of IEEE Software, a professor and doctoral supervisor in the Department of Computer Science and Technology, East China University of Science and Technology. His current research areas include software automation, CASE, software engineering management.

1 Requirements of Software Process Transaction

Software process is a sequence of steps, operating on software artifacts and coupled to some agents. They can be sequent, repeated, concurrent, nested, and iterated. Process model is the formalized description of process, which typically consists of function model, product model, resource model, and cooperation model[2]. In the E-Process system, a software process P may be defined as a 7-tuple:

$$P = (Ps, C1, C2, Is, Os, Rs, As).$$

Here, *Is* and *Os* are software products, such as programs and documents, which constitute the product model; *Ps* are sub-processes or operations, which constitute the function model; *C*1 is process pre-condition, and *C*2 is process post-condition. They constitute cooperation model; *Rs* are roles to execute process, such as programmer and compiler; *As* are agents, which are assigned to roles. They constitute resource model.

Software process is usually organized hierarchically. One process can be refined as a set of sub-processes. When pre-condition (*C*1) is satisfied, agents (*As*) can start the process, and generate a new process instance to execute process transaction; then they get input products (*Is*), and produce output products (*Os*); after post-condition (*C*2) becomes true, task can be finished, thereby the transaction is submitted. For such a transaction in this specialized environment, we might call it as software process transaction. Different from traditional transactions, software process transactions have the following characteristics:

(1) Interactive, dynamic, and long-duration

First, they are interactive and long-duration. A large part of their control is under the responsibility of human actors. Their execution is not predefined, but invoked by developers dynamically. This makes software development processes unpredictable, in the sense that their duration and effects are in many cases difficult to anticipate. Their duration can vary from a few days to months or even years. Also the individual actions executed by users, like editing, compilation, debugging, testing, etc., are usually of long duration.

Furthermore, real software process model can't be defined completely in early phases of software development. Usually, a process framework is established firstly before the project starts, then it is improved and refined continually during enactment. This evolutionary property enhances the mutability of software process transactions.

(2) Iterative

Software development processes are usually iterative and evolutionary. Such incremental process pattern is recommended, like rapid prototype. It allows to progressively converge to products of complex systems that hold the desired qualities, based on risk analysis. In each development phase, activities, like editing, compilation, testing and debugging, can also be repeated.

(3) Multi-User cooperation

Software development processes are jointly carried out by several users. For this, the software engineers need to exchange information during their work in a controlled way, and take profit of and influence the activities of other users working in parallel on the same resources.

(4) Product-Focused

Different from workflow applications, software artifacts and products are primary elements and final results of software processes. It is essential to control changing and tracing of products flexibly and effectively in software process transaction.

(5) Openness

The sharing and exchange of both final and intermediate results among cooperators is a prerequisite for software process transaction, while at the same time guaranteeing that no anomalies are introduced by the exchange.

Thus, software process transactions should support exposure of uncommitted data.

These key features of software process challenge PSEE to provide an effective transaction model, to facilitate cooperation and guarantee correctness.

2 Criteria-Based Software Process Transaction Model

Software processes pose new requirements to transaction management. In such environments, strict ACID properties (Atomicity, Consistency, Isolation and Durability) conflict with cooperation, interaction, evolutionary and openness features of software process. Software process transaction model should support developers, managers and QA staffs to exchange and share information naturally, while guaranteeing data consistent and correct. Our proposed criteria-based software transaction model E-Process/TM can meet these requirements quite well, as shown in Table 1.

Table 1 Software process supports in E-Process/TM

Key Features of E-Process/TM	Requirements of software process transaction
Dynamic hierarchy transaction structure	Interactive, Iterative
Process evolution	Dynamic
Exposure of uncommitted data	Cooperation, openness
Selectable compensation policy	Openness, interactive
Check-In/Check-Out	Cooperation, product-focused
User defined correctness criteria	Cooperation, openness, dynamic, and long-duration

In E-Process/TM model, the overall transaction structure is a tree corresponding to hierarchical process structure: process transactions, sub-process transactions and operation transactions. Here, (sub-)process transactions are no-atomic, and allowed to fail partially; while operation transactions follow ACID properties strictly, thus are atomic. Transactions are created by user dynamically, submitted or aborted after finishing all operations. Nested process definition introduces nested transactions. When the process is enacted in iterative approach, it will generate multi-transactions. If process is improved, consequently its transactions will be quite different. In this sense, transaction hierarchy is fully dynamic. E-Process/TM supports such uncertainty through its process evolution mechanism, instead of dynamic structuring in split transaction model[3,4].

To enhance process cooperation and parallelism, E-Process/TM adopts more open policy than nested and multi-level transaction[3,5,6], with exposure of uncommitted software products. That means, it allows not only exposure of sub-transaction data before parent is committed, but also exposure of uncommitted products in any transaction. The possibility of cascading rollback caused by failed transaction will be handled by its compensation transaction[7], erasing the effects. Sometimes it is necessary to retain activity history in spite of failure or not. For example, calculating workloads, measuring progress, and etc. Thus, E-Process/TM will interact with the user to determine the proper form of compensation.

Version management is the essential and indispensable activity in software development process. E-Process/TM integrates version management based on Check-In/Check-Out mechanism[9]. However guaranteeing configuration correctness is its key issue to make different data objects form a consistent set of versions. With well-defined correctness criteria, E-Process/TM provides a flexible concurrency control to deal with it finely. Before executing operations, transaction manager will verify them according to correctness criteria. These criteria are pre-defined based on the semantics of software process domain. They like a firewall to stop invalid operations. Compared with serializability, this approach enhances higher parallelism, and is suited to user-interactive, cooperation transaction with bigger granular data, such as software process.

Next two sections will introduce transaction definition and eight correctness criteria respectively in detail.

2.1 Definition of software process transaction

The main task of concurrency control is to ensure shared data consistent and correct. In the domain of software process, the shared data are software products. So, transaction can be abstracted as a finite set of product access operations. E-Process/TM extends operation primitives based on traditional transactions, defined as table 2. Because data operations on user private workspace don't effect overall situation, they aren't considered.

Table 2 Seven atomic operations in software process transaction

No	Operation primitives	Signs	Semantic
1	Read only	*Ro*	Copy products from public workspace to private workspace, and won't write back later.(Check-Out)
2	Read for writing	*Rw*	Copy products from public workspace to private workspace, and will write back later.(Check-Out)
3	Read for Reference	*Rr*	Copy products from public workspace to private workspace, just for reference. It is to say, not only products won't be written back, but also their modification has few effects on this task. (Check-Out)
4	Write	*W*	Copy created or changed products as a new version from private workspace to public workspace.(Check-In)
5	Prove	*P*	Copy products from public workspace to private workspace, take a technical review, then modify its meta-data in public workspace.
6	Commit	*C*	Transaction is finished successfully, and all of its products are committed.
7	Abort	*A*	Transaction is failed, and its effects are canceled by compensation transaction.

Therefore, in E-Process/TM, software process transaction $\boldsymbol{T}$ can be defined as:

$$T=\begin{cases}o_1o_2...o_n & \text{or}\\ \{t_1,t_2,...,t_n\}\end{cases}.$$

If software process transaction is a partial-order set of operations on software products: $o_1o_2\cdots o_n$, and their partial-order is $<$, then

I. $o_i \in \{ro_i(x), w_i(x), p_i(x), rw_i(x), rr_i(x) \mid x \text{ are products}\} \cup \{a_i, c_i\}$;

II. $a_i \in T_i$ iff $c_j \notin T_i$;

III. if $t = c_i$ or $t = a_i$, then $\forall(p \in T_i)\ p < t$, and

IV. if $r_i(x), w_i(x) \in T_i$, then $r_i(x) < w_i(x)$ or $w_i(x) < r_i(x)$.

Here, condition (Ⅰ) defines seven operation primitives in transactions; (Ⅱ) means that transactions must be committed or aborted exclusively; (Ⅲ) means that after transaction is submitted or aborted, it cannot take any operation; (Ⅳ) means that operations on the same product should follow the partial order.

Software process transaction also can be a finite set of sub-transactions. These sub-transactions are running in parallel under the restrictions of control flow and data flow between them.

2.2 Correctness criteria

There is no single correctness criterion like global consistency and atomicity that supports cooperation applications[3]. Rather, the notion of what is correct may vary from application to application and from task to task. Hence, we have to allow user-specified correctness criteria. So in the domain of software process enactment, we provide eight correctness criteria, as operation verification foundation.

Criterion 1. Allowing transaction T_i to expose uncommitted products to transaction T_j, i.e.

$$w_j(x) < r_i(x) < c_j \quad \text{or} \quad w_j(x) < r_i(x) < a_j,$$

here, $r \in \{rr, ro, rw\}$.

Criterion 2. If a product is written by transaction T_j, then other transactions can't write it until T_j is finished.

i.e.

$$\text{if } w_i(x) < w_j(x) \text{ , then } w_i(x) < c_i < w_j(x) \quad \text{or} \quad w_i(x) < a_i < w_j(x).$$

Criterion 3. If transaction T_i reads products from T_j, then T_i can't be committed before T_j is committed. i.e.

$$\text{if } w_j(x) < r_i(x),\ c_j \in T_j \text{ and } c_i \in T_i \text{ , then } c_j < c_i,$$

here, $r \in \{ro, rw\}$.

Criterion 4. If transaction T_i reads products from T_j before T_j is committed, and then T_j rewrite the products, then T_i must re-read the products. i.e.

$$\text{if } w_j(x) < r_i(x) < w_j'(x),\ c_j \in T_j \text{ and } c_i \in T_i \text{ , then } w_j(x) < r_i(x) < w_j'(x) < r_i'(x),$$

here, $r \in \{ro, rw\}$.

Criterion 5. If transaction T_j modifies the products read by T_i for reference, then just need to notify T_i operator. i.e.

$$\text{if } w_j(x) < rr_i(x) < w_j'(x),\ c_j \in T_j \text{ and } c_i \in T_i \text{ , then } w_j(x) < rr_i(x) < w_j'(x) < rr_i(x).As.Notify.$$

Criterion 6. If Transaction T is aborted, then its compensation transaction must be executed to cancel its effects. i.e.

$$\text{if } a \in T, \text{ then } a < T^-,$$

here, T^- is the compensation transaction of T.

Criterion 7. Products must pass the technical review before transaction is committed. i.e.

$$\forall w_i(x), \text{ if } c_i \in T_i \text{ then } \exists\ T_j, (w_i(x) < p_j(x) < c_i \text{ and } p_j(x) = \text{True}).$$

Criterion 8. If transaction T is an instance of process P, then the execution of T must be consistent with P. It is to say, T can only read input-products of P (Is), and write output-products of P (Os). i.e.

$$\text{if } T \text{ is an instance of } P, \text{ then } (\forall w(x)\ , x \in Os) \text{ and } (\forall r(x)\ , x \in Is),$$

here, $r \in \{ro, rw, rr\}$.

Skarra[11] and Nodine[3] defined a correctness specification in terms of conflict and pattern. Conflicts identify operations that are not allowed to execute concurrently, such as write lock; while patterns define sequences of operations that must occur, such as atomicity property. Thus, one transaction is correct if and only if it follows all patterns, and excludes any conflict. Table 3 classifies eight correctness criteria in these terms.

Table 3 Correctness criteria classification

Properties	Correctness criteria	Pattern/Conflict
	Criterion 1	Pattern
Quasi-Isolation	Criterion 2	Conflict
Quasi-Atomicity	Criterion 3	Conflict
	Criterion 4	Pattern
	Criterion 5	Pattern
Recoverability	Criterion 6	Pattern
Quality assurance	Criterion 7	Pattern
Consistency with process	Criterion 8	Pattern

3 Transaction Model Implementation

Currently, no commercial or academic results of nested cooperative transaction models have reached a status stable enough for commercial implementation[2]. At the same time, some available PSEE systems, such as famous SPADE[2], don't offer any primitive mechanism for long or nested transactions. Users should program to ensure software process transaction consistent by themselves. E-Process integrates criteria-based software process transaction model E-Process/TM, assisting effective and cooperative process enactment. Thus, process engineering can focus on process logic in high level, not have to pay attention to transaction logic in detail. The architecture of

E-Process is based on a multi-tier client/server architecture, as shown in Fig.1, to take full advantages of web application server technologies. On the client side, E-Process executes graphical user interface logic through web browser; on the server side, web server, Java application server and database server constitute the multi-servers structure. Thus, E-Process is basically partitioned into two working parts: E-Process client and E-Process server (i.e. process engine). The whole system is implemented by J2EE programming model. Using Java applet, servlet and Entity Java Bean, it supports separation of UI logic, business logic and data logic, therefore, is scalable and open.

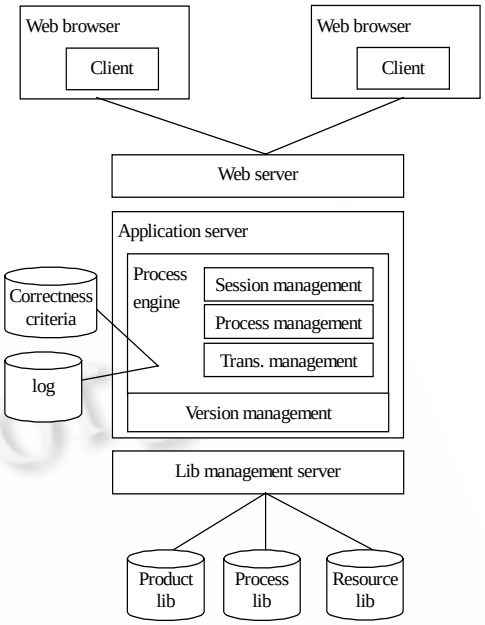

Fig.1 E-Process architecture

Application server is the core of E-Process system. It consists of four key components:

(1) session management component: manage state data in user sessions;

(2) process management component: cooperate multi-user development activities;

(3) transaction management component: guarantee correctness and data consistency of software process transaction, based on version management;

(4) version management component: implement product Check-In/Check-Out model.

The first three components compose process engine. Obviously, E-Process/TM is implemented by transaction management and version management components, to

(1) manage the whole life cycle of software process transaction, through creating, product accessing, committing, to aborting and compensating;

(2) adjust transaction hierarchy dynamically;

(3) verify operations according to rules in the correctness criteria library;

(4) exchange products between private and public workspace by check-in and check-out operations, and

(5) record all operations in log library, which are used during compensating when transaction is failed.

The correctness criteria of software process transactions are defined in logic expressions. Eight built-in correctness rules have been stored in library. They allow process experts to customize and expend, even to define a new suit.

4 Related Work

Transaction is the key concept of database management. In traditional transaction model, each of the concurrency control protocols, such as 2PL and timestamp, has adverse effects on long duration transactions[2,3]. It appears that strict ACID properties limit cooperation and decrease performance. Thus, several advanced transaction models have been proposed in recent years.

(1) Nested and multi-level transaction

Nested transaction was first introduced by Moss in 1985[5], which decomposes a long transaction into a number of sub-transactions. By enhancement of flat to nested transactions, we are able to execute sub-transactions in parallel, and finely control over failures and errors, thus achieving recovery guided by semantic information. In the nested transaction model, when a child transaction finishes, locks are inherited by the parent. If locks are simply

released, it is called multi-level transaction[3,6]. When a multi-level transaction represents a long-lived activity, sometimes it is called as Saga[7].

(2) Compensation transaction

To reduce the frequency of long-duration waiting, we arrange for uncommitted updates to be exposed to other concurrently executing transactions. For example, multi-level transactions allow the exposure of sub-transaction data before parent is committed. However, the exposure of uncommitted data creates the potential for cascading rollbacks. The concept of compensation transaction helps us to deal with this problem, based on the semantic information[7]. In this model, every transaction has its compensation transaction. When it is aborted, the compensation transaction will be started to recover data to previous consistent state, instead of cascading rollback.

(3) Split transaction

Split transaction[3,4] proposes a solution for restructuring in-progress transactions. It supports dynamics, open-ended, multi-users cooperative activities. In real applications, split transactions are a natural means of committing some work early or dividing on-going work among several cooperators. Advantages of this model are adapting recovery effort and reducing isolation.

(4) Cooperative transaction

The concept of cooperative transaction was introduced by Nodine in 1992[3], to support a higher level cooperation. This model contributes to its user defined transaction correctness criteria, substituting the notion of correctness defined by serializability. Further, hierarchy structure allows easier management for long-lived transaction.

(5) Participant transaction

The participant transaction model defines each transaction as a participant in a specified domain[8]. The domain represents the set of user transactions controlled by users collaborating on a common task. Participant transactions in the same domain need not be serializable—only transactions of different domains have to be serializable.

(6) Check-Out/Check-In model

One of the simplest forms of synchronizing the access of a team of designers to a shared repository is the check-out/check-in model[9]. This model has been implemented by widely used version control and configuration management tools. Data objects have to be explicitly checked-out by a designer, i.e., objects are copied from the shared database to the private workspace, where the data objects can be manipulated. In the simplest case, checked-out data objects are reserved for exclusive access, until they are later checked-in into the shared database. In another case, data objects are allowed to check out by multi-users at the same time. This results in branches in the version history of an object. For the purpose of software development, several extensions of the check-out/check-in model have been developed, which take advantage of the possibility of software consistency checking in those environments. For example, introducing new operations, locking schemes and semi-public workspace[10]. COO system developed by ECOO team takes this approach[2].

(7) Transactional workflow

Workflow addresses these issues too. Interesting transactional workflow approaches are, ConTract model, ATM, Flex transaction, ACTA, S-transaction, Polytransaction, TAM, WIDE, TransCoop, and etc.

The advanced transactions models presented above provide good support for some cooperative applications. However, software process transactions pose particular requirements. They fail to meet all of them:

(1) In nested and multi-level transaction model, sharing of tentative data among co-workers executing concurrent transactions is not possible because both of them stick to a variant of serializability as their correctness criteria.

(2) Compensation transactions only focus on transaction recovery mechanism, and cooperative transaction is

complex and general.

(3) The open problem in split transaction model, is how to ensure that two transactions resulting from a split form again a consistent unit of work.

(4) As the participant transaction model is based on read/write actions and no concurrency control applies to a domain, this can lead to inconsistencies of data accessed in a domain and counter-intuitive behavior of the system from the user' s viewpoint. Moreover, support for relaxed atomicity, user-initiated revision of work, and for private and shared data is limited.

(5) In Check-Out/Check-In model, the parallel versions have to be merged manually in order to come up with a single, commonly agreed result of the object. Another problem is to keep track of which versions of different data objects form a consistent set of versions. Dealing with configurations complicates version management and check-out/check-in. In addition, this can decrease concurrency because entire configurations have to be reserved always.

(6) The focus of all transactional workflow approaches is on coordination, i.e., the coordinated execution of multiple related tasks. Flexible but controlled sharing and exchange of data objects among co-workers participating in the same task or in different tasks is not in the focus of transactional workflow approaches. This is essential in software development.

5 Conclusions

Transaction management mechanism is the key component of process-sensitive engineering environments. Based on domain semantics, we introduce good concepts and policies from cooperative transaction, compensation transaction and Check-in/Check-out mechanism, propose a criteria-based software process transaction model E-Process/TM, and implement it in commercial E-Process system. This model can meet requirements of software process well, and presents seven main important advantages:

(1) dynamic transaction hierarchy
(2) execution of iterative processes
(3) extension of operation primitives to reflect software process peculiarity
(4) exposure of uncommitted products
(5) selectable compensation policy
(6) version management integration
(7) user defined correctness criteria

Since E-Process V1.1 was developed, we launched some pilot projects firstly in ASTI Shanghai, and aided the company to get ISO9001 certification successfully on December 2000. Future work will focus on distributed collaborative process modeling, which has some different aspects.

Acknowledgement We would like to thank K. Kishida and Y. Matsumura of SRA Japan for providing valuable comments. We also acknowledge cooperation and supports of all members of ASTI.

References:

[1] Shen, B.J., Gu, C.H., Chen, C., *et al*. A distributed architecture for process-sensitive engineering environments. In: Proceedings of the Conference on Software: Theory and Practice, the 16th World Computer Conference. Beijing: Publishing House of Electronics Industry, 2000. 993~994.

[2] Derniame, J., Kaba, B.A., Wastell, D. Software process: principles, methodology, and technology. Heidelberg, Germany: Springer-Verlag, 1999.

[3] Elmagarmid A. Database transaction models for advanced applications. San Francisco, CA: Morgan Kaufmann Publishers, Inc.,. 1992.
[4] C. Pu, G. E. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In: Proceedings of the 14th International Conference on Very Large Databases (VLDB). Los Angeles, CA: Morgan Kaufmann Publishers, Inc., 1988. 26~37.
[5] Moss, J. Nested transactions: an approach to reliable distributed computing. Cambridge, Massachusetts: MIT Press, 1985.
[6] Weikum, G. A theoretical foundation of multi-level concurrency control. In: Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. Cambridge, Massachusetts: ACM Press, 1986. 31~42.
[7] Garcia-Molina, H. and Salem, K. Sagas. In: Proceedings of the ACM SIGMOD Conference. San Francisco, CA: ACM Press, 1987. 249~259.
[8] Kaiser, G. E. MARVEL 3.1: A multi-user software development environment. In: Proceedings of the International Symposium on Logic Programming. Vancouver, Canada: MIT Press, 1993. 36~39.
[9] Rochkind, M. The source code control system. IEEE Transactions on Software Engineering, 1975,1(1):364~370.
[10] Kim, W., Lorie, A., McNabb, D. and Plouffe, W. A transaction mechanism for engineering design databases. In: Proceedings of the 10th International Conference on Very Large Databases (VLDB). Singapore: Morgan Kaufmann Publishers, Inc., 1984. 355~362.
[11] Skarra, A. H. Concurrency control for cooperating transactions in an object-oriented database. ACM SIGPLAN Notices, 1989, 24(4):145~147.
[12] Mo, Q., Li, Z. M., Tan, Y.S., *et al*. Flexible cooperative transaction model. Journal of Software, 2000,11(6):720~726 (in Chinese).

附中文参考文献：
[12] 莫倩,李子森,谭郁松,等.柔性协同事务模型.软件学报,2000,11(6):720~726.

基于规则的软件过程事务模型

沈备军[1,2], 陈 诚[2], 居德华[1,2]

[1](华东理工大学 计算机系,上海 200237);
[2](中国科学院 软件研究所,北京 100080)

摘要: 事务管理是过程工程环境（PSEE）的一个关键技术.近年来,提出了不少高级事务模型支持长事务.然而,由于软件过程事务处理的特殊性,这些模型只能反映其中一部分需求,而且大多数多层的合作事务模型还不够稳定,无法应用于商业,这对 PSEE 的事务机制提出了挑战.基于此,提出了一个基于规则的软件过程事务模型 E-Process/TM,能够较好地刻画软件过程的特征,即用户交互性、长周期、迭代式过程和数据共享的多用户协作.基于用户可自定义的正确性规则,E-Process/TM 提供了良好的灵活性和开放性.目前,该模型已在商业 PSEE 产品中得以实现,并成功地应用于实践中.

关键词: 软件过程;软件过程事务;扩充事务模型;软件过程支持环境;正确性规则
中图法分类号: TP311 文献标识码: A