NPS52-90-018

NAVAL POSTGRADUATE SCHOOL

Monterey, California

FLEXIBLE COOPERATION IN
NON-STANDARD APPLICATION ENVIRONMENTS

Bernhard Holtkamp

January 1990

Prepared for:

Naval Postgraduate School
Monterey, California 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared for the Naval Ocean Systems Center and funded by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:

BERNHARD HOLTKAMP
Adjunct Research Professor
Department of Computer Science

Reviewed by:

ROBERT B. MCGHEE
Chairman
Department of Computer Science

Released by:

G. SCHACHER
Dean of Faculty
& Graduate Studies

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): NPS52-90-018	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): 52	7a. NAME OF MONITORING ORGANIZATION: Naval Ocean Systems Center
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code): San Diego, CA 92152
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: O&MN, Direct Funding
8c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
FLEXIBLE COOPERATION IN NON-STANDARD APPLICATION ENVIRONMENTS

12. PERSONAL AUTHOR(S)
Holtkamp, Bernhard

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Progress	FROM Ap 89 TO Dec 89	1990 January	27

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Cooperative systems, database integration, transaction concepts, decentralized systems, S-transactions

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The integration of preexisting systems into a single, heterogeneous, distributed non-standard application system in domains like office automation or computer-integrated manufacturing are regarded as cooperating systems. They are characterized through teamwork, distribution and the handling of complex data structures (e.g. multimedia data). Object-oriented database systems, providing for complex object management, represent one approach in support of such applications. They concentrate, however, on data modeling aspects and use more or less conventional transaction concepts, based on a global execution control. Hence, they only partially fulfill application requirements as they do not adequately cope with the autonomy that is often inherent to the system's components.

As a consequence, we suggest S-transactions as an appropriate means for describing the cooperation of system components in terms of transactions and beyond. In this paper we outline the modeling of conventional transactions (flat or nested as well as distributed and design transactions) in terms of STDL, the S-transaction definition language. Beyond that we point out how to specify SAGAs and similar concepts. Finally we

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Bernhard Holtkamp	22b. TELEPHONE (Include Area Code): (408) 646-2251; 22c. OFFICE SYMBOL: 52Hk

discuss the specification of non-linear but maybe acyclic or even cyclic cooperation structures.

FLEXIBLE COOPERATION

IN

NON-STANDARD APPLICATION ENVIRONMENTS

Bernhard HOLTKAMP*)

Naval Postgraduate School
Computer Science Department
Monterey, CA 93943
USA

E-Mail: holtkamp@cs.nps.navy.mil

Abstract

The integration of preexisting systems into a single, heterogeneous, distributed non-standard application system in domains like office automation or computer-integrated manufacturing are regarded as cooperating systems. They are characterized through teamwork, distribution and the handling of complex data structures (e.g. multimedia data). Object-oriented database systems, providing for complex object management, represent one approach in support of such applications. They concentrate, however, on data modeling aspects and use more or less conventional transaction concepts, based on a global execution control. Hence, they only partially fulfill application requirements as they do not adequately cope with the autonomy that is often inherent to the system's components.

As a consequence, we suggest S-transactions as an appropriate means for describing the cooperation of system components in terms of transactions and beyond. In this paper we outline the modeling of conventional transactions (flat or nested as well as distributed and design transactions) in terms of STDL, the S-transaction definition language. Beyond that we point out how to specify SAGAs and similar concepts. Finally we discuss the specification of non-linear but maybe acyclic or even cyclic cooperation structures.

*) The author's permanent address is: University of Dortmund, Computer Science Department, Software Technology, P.O. Box 500 500, D-4600 Dortmund 50, West Germany, E-Mail: holtkamp@stltar.informatik.uni-dortmund.de

This work was performed while the author was with the Multimedia Database Project at the Naval Postgraduate School. The work was in part sponsored by the Naval Oceans System Center (NOSC) as project no. RC32510 and is strongly influenced by the work done in the MAP project 761/761B, sponsored by the Commission of the European Communities.

1.0 INTRODUCTION

Non-standard application domains like office automation and computer integrated manufacturing are characterized through complex collaboration patterns of people and complex interrelations of organizational units. A further characteristic property is the management of or at least the demand for multimedia documents, i.e. documents that integrate text, images and graphics and eventually even sound. Supposed that such multimedia documents are kept in appropriate database systems, the management of these documents is supposedly done by means of transactions in the sense of database transactions (c.f. [Gray78]). The development of extended relational database systems (e.g. POSTGRES [SR86]) and object-oriented database systems (OODBMSs) for the management of multimedia documents provide first solutions [KiLo89]. Although these systems are based on advanced data models, the provided transaction mechanisms are more or less conventional.

However, it has been pointed out earlier that the conventional transaction concept is not adequate for advanced applications [Gray81]. Consequently, the transaction concept has been revised in several respects. First of all the traditional concept has been extended towards distributed systems (c.f. [CePe84]). In order to increase parallelism within a transaction the concept of nested transactions [Moss81] has been invented. In [Gray81] it is outlined that conventional transactions are inappropriate for long-lasting activities such as design applications, for instance. To close this gap, engineering design transactions of different flavors have been developed [Kelter88]. While all these concepts are bound to a global execution control, most recently concepts have been developed [GaSa87], [ElVe88], [Holtkamp88], [Johannsen89] [ISO88] that enable the autonomy of sites in such environments. This autonomy might occur in different respects, [GaKo88], [Gray86], [Wshop88], [ElVe88].

Simple examples immediately illustrate, that there is a strong demand for the coexistence of several if not all of the above mentioned transaction concepts in one system, supporting non-standard applications. In other words, the integration of data into complex structures (e.g. a design document consisting of text graphics and images) and the integration of systems (e.g. a multidatabase system or federated database system) into a single, more complex one also implies the integration of cooperation principles.

A query of the type "Who is the author of report XY?" can easily be executed as a conventional transaction. The collection of contributions to a common report from different groups can be mapped into a distributed or nested transaction, depending on the organizational relations between the groups. The development of a design document is modeled best by means of a design transaction. The collaboration of different more or less autonomous groups (e.g. request of a marketing group to the design group "Give us a drawing of machine ABC for a marketing brochure") demands more liberal concepts.

In the sequel of this paper we demonstrate that the concept of S-transactions [MAP88a] is well-suited to cover all requirements sketched above, i.e. S-transactions support the integration of the different transaction concepts and enable execution patterns that go beyond those.

In order to make the mappings of the different transaction models into S-transactions understandable we start with a brief review of the S-transaction concept and the underlying

system model. Based on this review we model conventional "flat" transactions, simple and nested distributed, and design transactions by means of S-transactions. The fourth section outlines the use of S-transactions for advanced transaction structures like "triangles", "cycles" or acyclic graph structures, different commit protocols and the delegation of control and coordination functions among S-transactions.

2.0 S-TRANSACTIONS AND THE UNDERLYING SYSTEM MODEL

The concept of S-transactions has originally been developed for transnational banking applications where banks cooperate but preserve their autonomy [MAP88aj. As a consequence of the strong demand for autonomy, conventional transaction mechanisms are not applicable (c.f. [Holtkamp88]). Therefore, S-transactions and the corresponding definition language STDL have been introduced in order to describe the cooperation of autonomous components in an integrated system.

In the sequel of this section we highlight S-transaction features, STDL and the underlying system architecture model in order to ease the understanding of why and how S-transactions can be used for modeling different kinds of transactions.

2.1 S-Transactions and STDL

S-Transaction Concept

An S-transaction ST is recursively defined. It (i.e. the global S-transaction) consists of a *partially ordered set of subordinate S-transactions and local transactions* (LTs). The local transactions interface to the applications that are allocated to the local sites.

The implementation of the local transactions is in the responsibility of the local sites (due to their autonomy, c.f. [Holtkamp88]). Hence, implementations of the same LT may differ from site to site. Its semantics, however, must be the same at each site. Local transactions are supposed to be executed as transactions in the classical sense (as described in [Gray78], for instance,) exhibiting ACID properties*). This assumption is based on the assumption that the local databases are conventional databases of a relational, hierarchical or network type. An LT is executed under the control of the local site and the site is responsible for the recovery of the failing transaction. Thus, we preserve the local site's execution autonomy as the LT execution control is beyond the scope of the ST that calls the LT.

Recovery of S-transactions is based on compensation [Gray81]. For any component S-transaction ST and for any local transaction LT we require the existence of compensating transactions -STs and -LTs, respectively. That corresponds to the SAGA approach [GaSa87], where for each transaction that forms a part of a SAGA, the existence of a compensating transaction is required, too.

---

*)ACID stands for atomicity, consistency, isolation and durability. Atomicity means that a transaction is either completely executed or not at all. Consistency means that a transaction transfers a database from one consistent state into another. Isolation means that a concurrently running transaction does not get access to data that are used by another transaction prior to the termination of the latter. Durability means that once a transaction has committed, its effects are persistent and are not affected by a system failure, for instance.

S-Transaction Features

As the above definition shows, S-transactions do not provide ACID properties in the sense of conventional transactions.

Atomicity is only available from the semantic point of view. Recovery is based on compensation, i.e. the component transactions have alredy committed prior to the end of the global S-transaction. Thus an UNDO by inserting a before image that has been stored in a log file is not possible as concurrent transactions might have already modified the data objects in the mean time. Instead, a compensating transaction is executed that reverses the effects of the previous one in the sense of the application.
This view of semantic atomicity directly relates to consistency, i.e. S-transactions cannot provide consistency in the traditional sense. S-transactions can only lead to some state that is, from the application point of view, "consistent enough" as described in [Garcia83].
S-transactions are isolated against each other in that no S-transaction can access data, defined in a concurrent one. Local transactions are conventional transactions and thus also provide for isolation during their lifetime. As they commit however prior to its superior S-transaction, isolation is given up at this point, thus allowing for all parallelism anomalies like "dirty" reads, lost updates and unrepeatable reads.
S-transactions are durable in the conventional sense, i.e. once an ST has terminated its effects can only be reversed by executing a reverse transaction.

In contrast to the global execution control of conventional approaches S-transactions are based on decentralized control, i.e. control migrates or at least can migrate from site to site.

S-Transaction Type Description

An S-transaction type is described in STDL according to the schema depicted in figure 2-1.

```
s_transaction <name>
input_data
    <set of input data>
end
result_data
    <set of requested messages>
end
private_local_data
    <set of private local data>
end
continuation_points
<set of continuation points>
end
end /* of S-transaction <name> */
```

Figure 2-1: Schema of an S-transaction definition

The input data section contains the data type definitions for the input parameters that have to be filled in when invoking an S-transaction. The result data section contains the data type definitions for the data that are returned as results to the initiator of an S-

transaction. The private local data section contains the data type definitions for the data that are needed by the S-transaction internally during its execution. The continuation points denote the entries at which an S-transaction may be activated or may receive results. The special continuation point 'init_cp' is activated at S-transaction initiation time if the initiation is requested from a user. The other continuation points can be activated either from remote sites by requesting a service, provided by that S-transaction type the continuation point belongs to, or by submitting results that were requested from another continuation point of the already active S-transaction.

A continuation point (CP) consists of a head, similar to the signature of a procedure, and a path, corresponding to a procedure's body. The head is composed of the CP's name and a formal parameter list. A path can be formed of LT calls, requests of services from remote sites, result deliveries to remote sites, thus providing a previously requested service, acknowledgments, confirming the receipt of a message or the execution of a requested service, abort notifications in case of local failures and standard arithmetic, boolean and set operations. All these operations are separated from each other by means of control flow determining operators like sequence (';') conditional execution ('IF-THEN-ELSE' or 'CASE') or iteration ('FOR' or 'WHILE'), well known from conventional programming languages.

For a detailed description of STDL we refer to [HaHo87] or [MAP88a].

2.2 System Model

The S-transaction concept underlying system model is a set of cooperating autonomous sites. Each site has its own local database system (LDB) and interfaces to the software part that provides the integration into the cooperative system (fig. 2-2).

The integration component (bold frame) consists of the S-Transaction Management (STM), a communication subsystem (Com), an interface to the local application (LAI) and a user interface (UI).

The STM controls the execution of S-transactions and coordinates the execution of STs and LTs. It provides time concept for the definition of time-oriented triggers (e.g. time-out). All actions of the STM are performed as conventional transactions, i.e. they are robust against system failures. The STM also supports a log-based recovery mechanism for S-transactions.

The local application interface LAI implements the communication between the integration software layer and the local application program. Depending on the actual implementation this communication can range between procedure calls, interprocess communication and communication over a local area network.

The communication subsystem has basically been designed for an OSI-based store & forward communication discipline (i.e. X.400 based [MAP88b]) but it has been shown that it can easily be extended towards on-line communication, too [Becker89].

The user interface UI provides for initiation, status control and result management of S-transactions. As S-transactions are predefined UI is forms-oriented, i.e. the user has to fill-out a form for each desired service.

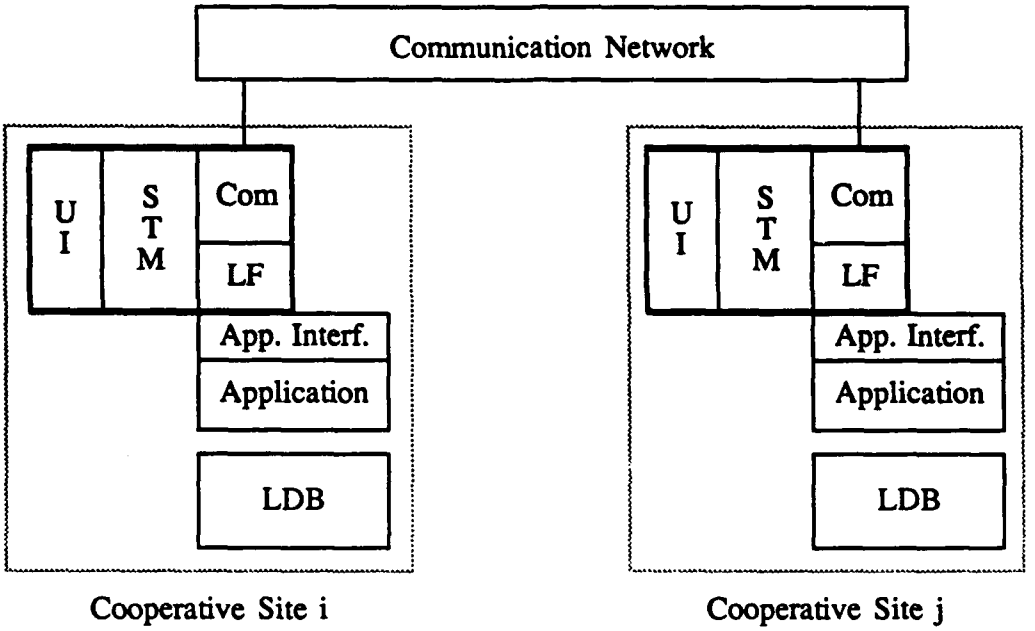

Figure 2-2: Model of a cooperative system

3.0 S-TRANSACTION TEMPLATES FOR CONVENTIONAL TRANSACTIONS

In the sequel of this section we discuss the modeling of conventional transactions as known from centralized database systems, of distributed either flat structured or nested transactions, of engineering design transactions and of SAGAs by means of S-transactions. Each approach is briefly outlined and then the corresponding S-transaction template is described in detail.

3.1 Conventional "Flat" Transactions

Concept Overview

According to the conventional transaction concept as it has been developed for centralized database systems a transaction consists of an action sequence that is embraced by keywords that indicate the beginning and the end of a transaction (TA):

TA_BEGIN
a_1
a_2
...
a_n
TA_END

Figure 3-1: Conventional transaction structure

The actions a_i represent read or write operations on the database. For the entire transaction ACID properties are guaranteed.

S-Transaction Template

According to the definition of S-transactions in section 2 a local function LT is automatically executed as a transaction in the above sense. Thus, either the entire action sequence has to be passed on to the local system as a parameter of the local transaction request or the interface to the local site has to be modified. As the first approach does not necessarily work (e.g. some intermediate operations are performed in-between two actions) only the second approach provides a solution. Hence, in order to enable the modeling of conventional transactions by means of S-transactions a slight modification of the concept is necessary.

A straightforward solution is the introduction of two different types of local functions. The one is the already existing LT type, i.e. the requested service is directly executed as a conventional transaction. The second type is a normal function call like a remote procedure call, for instance. We require the export of functions from the local site for transaction begin, transaction commit, transaction abort, lock request and lock release.

As the local site is supposed to execute conventional transactions, the export of the primitives listed above is not a major problem but a simple extension that can easily be implemented.

The basic template for an S-transaction that represents a conventional flat transaction looks as follows.

```
S_Transaction conv_transaction
< data section >
continuation points
   init_cp[...]:==
            LF("TA_BEGIN", LTID into NIL) ;
            LF("a1", LTID, ...);
            LF("a2", LTID, ...);
            ...
            LF("an", LTID, ...);
            LF("TA_END", LTID into NIL)
   end /* init_cp */
end /* continuation points */
end /* conv_transaction */
```

Figure 3-2: S-transaction template for conventional transactions

Thus, it is evident that the structure of the S-transaction template corresponds exactly to that of the conventional transaction. The only difference is the introduction of a transaction

identification (LTID) as the operations at the local site are decoupled from the S-transaction and the LTID is necessary for establishing a link.

Another difference might arise regarding the actions within the transaction. The basic philosophy of S-transactions is the predefinition of services, i.e. an S-transaction can only request services that are exported from another instance (i.e. local or remote site). As a consequence, either the predefined set of actions a_1, a_2, ..., a_n or a generic function has to be exported by the local site. Supposed that the site runs a relational DBMS with an SQL interface a set of generic functions like "SELECT", "INSERT", "DELETE", "UPDATE" can be provided where the "FROM" and "WHERE" clauses are passed as parameters; or even more flexible, a function "SQL" and the entire SQL-statement is passed as a parameter.

3.2 Distributed Transactions

Concept Overview

The model of a distributed transaction (DTA) is the following. A global transaction (GTA) initiates subordinate transactions (SubTAs) at remote sites. The remote sites execute the SubTAs as conventional transactions as discussed in the preceding section. However, the final commit or abort of the SubTAs is coordinated by the GTA. In order to guarantee atomicity for a DTA the SubTAs write additional log information into their local logs in order to enable recovery from communication or site failures. The coordination of the DTA termination (commit or abort) is based on a commit protocol, basically 2-Phase-Commit.

The necessary activities are performed by the Global Transaction Manager (GTM) at the initiating site and Local Transaction Managers (LTMs) at the remote sites that perform the SubTAs.

The structure of a distributed transaction is illustrated in figure 3-3. For a detailed discussion of distributed transactions we refer to [CePe84], for instance.

S-Transaction Template

In order to enable the modeling of DTAs as S-transactions the same extensions are required as for the modeling of flat transactions. The introduction of a commit protocol does not affect the local sites but is handled on ST level.

The template of an S-transaction that implements a distributed transaction is described in figure 3-4.

It is worth noting here, that the STM of the initiating site acts as the coordinator of the DTA. Without loss of generality we can assume that all accesses to and operations on application data are performed at the leaf nodes within conventional transactions. The GTM_STM only passes data in-between Sub_TAs and controls the order of execution. At the end of the DTA the GTM_STM controls the execution of the commit protocol.

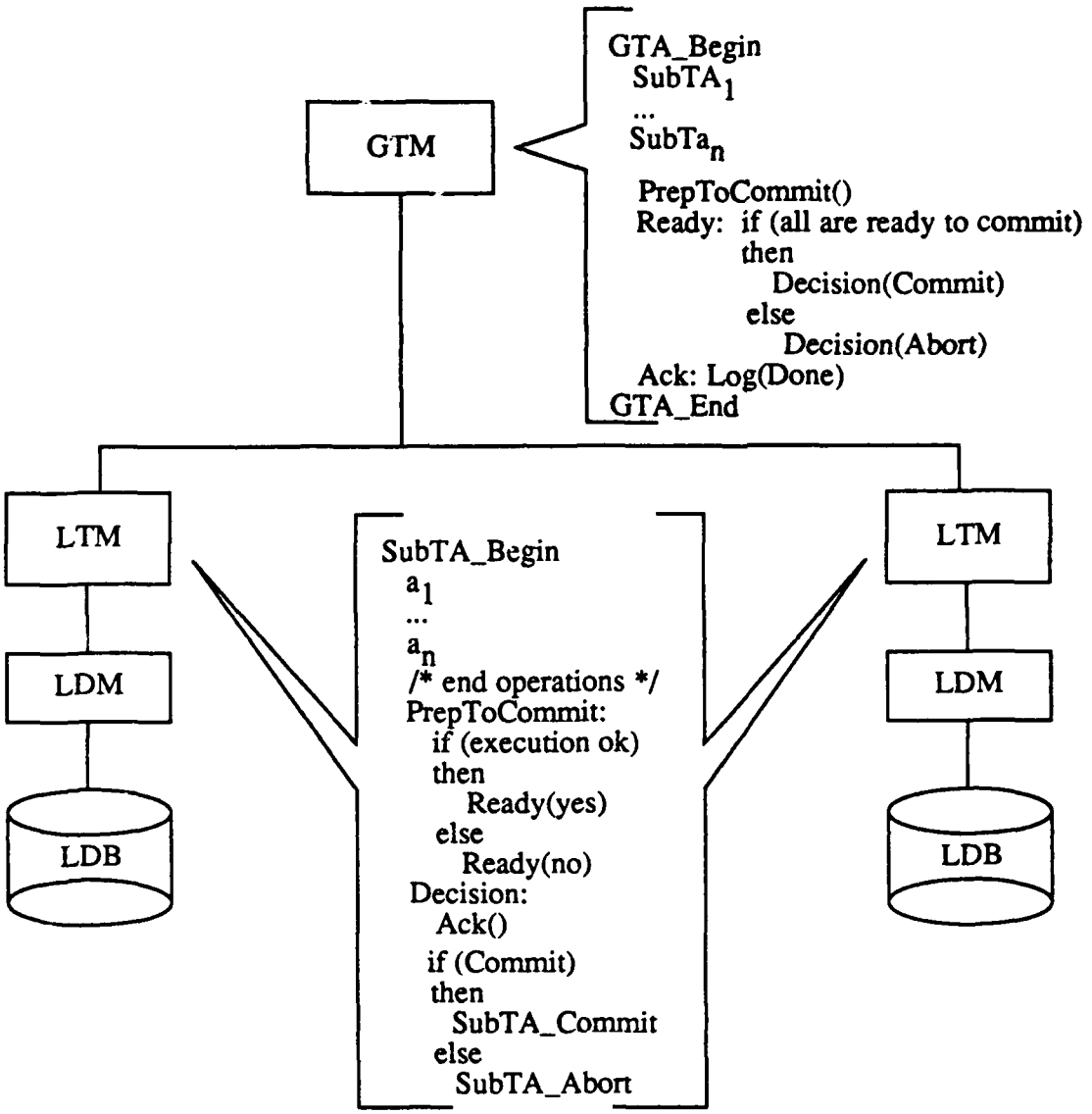

Figure 3-3: Structure of a distributed transaction

As a consequence, the GTM_STM needs only to protect the result data that are shipped from the remote sites, against accesses from outside. LTM_STMs must protect the data they receive from the local sites. In addition, all STMs must be stable with respect to system failures.

```
S_Transaction DTA
< data section >
continuation points
init_cp[...]:==  /* GTM at the initiating site requests subtransactions from remote sites */
                 request(site_i, STID, DTA, Sub_TA, ...); /* i = 1, .., n */
end /* init_cp */
aborted_cp[...]:== /* GTM receives an abort notification from a remote site and aborts the entire DTA */
                 abort(site_i, STID); /* i = 1, ..., n */
end /* abort_cp */
result_cp[...] :== /* GTM receives results from all involved sites; result_cp is activated
                                  when all results are available GTM asks sites to prepare for commit */
                 request(site_i, STID, DTA, prepare_cp, ...) /* i = 1, .. n */
end /* result_cp */
ready_cp[...]:== /* GTM receives ready_to_commit from all sites and decides on commit */
                 request(site_i, STID, DTA, commit_cp, ...) /* i = 1, .., n */
end /* ready_cp */
Sub_TA[...] :== /* STM of a remote site receives request for a subtransaction */
                 < processing of parameterlist received from GTA >
                 LF("TA_BEGIN", LTID, ...) ; /* initiation of the subtransaction at local site i */
                                 LF("a_ij", LTID, ...INTO SUCCESSFUL); /* and execution of actions a_ij
                                                                  with i = 1, ...,n and j = 1, .., mi */
                 if (NOT SUCCESSFUL)
                 then /* error has occurred during the execution of the transaction at the local
                                  site and the site indicates the abortion to the GTM */
                                 request(GTM_site, STID, DTA, abort_cp);
                 else /* local transaction has been executed properly */
                                 result(GTM_site, STID, DTA, result_cp, result_data);
                 fi
end /* Sub_TA */
prepare_cp[...] :== /* GTM asks for ready_to_commit notification */
                 result(GTM_site, STID, DTA, ready_cp, ready_signal)
end /* prepare_cp */
commit_cp[...] :== /* GMT decides on commit, now the local site can terminate the local transaction */
                                 LF("TA_END", LTID);
end /*commit_cp */
abort_cp[...] :== /* GMT has decided on abort, now the local site must rollback the local transaction */
                                 LF("TA_ABORT", LTID);
end /*abort_cp */
end /* continuation points */
end /* DTA */
```

Figure 3-4: S-transaction template for distributed transactions

3.3 Nested Transactions

Concept Overview

In order to increase the transaction internal parallelism the transaction concept has been extended towards nested transactions (NTAs) [Moss81].

In a graphical representation, NTAs are represented as trees. The root or top level transaction (TLTA) initiates subordinate transactions that might execute at the same site or at remote sites (i.e. distributed nested transactions). The subordinate transactions, in turn, can again initiate subordinate transactions again.

We distinguish between open and closed nested transactions [Johannson89]. Within closed nested transactions, the subordinate transactions preserve ACID properties against each other. The parent transactions inherit and preserve the locks of their child transactions after those have (pre-)committed. In open nested transactions [Traiger83], in contrast, child transactions release their locks after their commit in that way that other child transactions of the same parent transaction get access to these data but no transactions from outside.

S-Transaction Template

Like for the modeling of distributed transactions we require that the operations on application data are performed within the leaf nodes of the transaction tree.

The modeling of nested transactions by means of S-transactions is now straightforward. Per definition STs can be nested, i.e. they allow for tree structures. The operations on application data are performed within LTs.

In closed nested transactions, the LTs simply keep their locks until the entire NTA finally commits or aborts, respectively, i.e. the ST that is the parent of the LT delays the LT_END operation until it receives the corresponding request from its superior node. Each LT has its own identification. The locks on the local data are attributed with this identification.

As is obvious, the ST Management (STM) is freed from concurrency control with respect to the execution of LTs. Concurrency control and recovery for LTs are provided by the local DBMS. The local DBMS does not distinguish between LTs that belong to the same ST and those that belong to different ones. This has some implications on the modeling of open nested transactions.

If the local DBMS does not support open nested transactions, LTs are isolated from each other. Consequently, the parallel execution of child transactions must be performed as operations of the same LT. The separation of the child TA operations must be performed by the ST that is the parent of these child TAs. That means that in case of failure of one child transaction the ST cannot simply abort the entire LT. Instead, the STM must initiate the execution of compensating operations for the failing child TA in order to keep the other child TAs running that are performed within the same LT.

When implementing nested TAs by STs we have to distinguish between open and closed nested TAs, too. Closed nested TAs differ from distributed TAs only in that respect that SubTAs in a nested transaction are allowed to initiate further SubTAs. Thus the ST template for closed nested transactions is basically the same as for DTAs. In case of open nested

TAs the template is more complex as here the STM is to some extent in charge of concurrency control. We suggest to use sets of continuation points, each modeling a child TA of the parent ST. The local function calls of such a cp set correspond to the operations of the child transaction and their roll-back counterparts, respectively. Each set maintains a list of locks that it keeps on local DB data. Each set also has its own identification that is passed on to the local DB as a parameter of each LF call. Thus it is possible to retrieve the necessary compensation information from the log that is maintained by each STM in order to provide a basis for recovery.

3.4 Design Transactions

Concept Overview

The transfer of the "standard" transaction concept to the engineering design domain which is considered as a non-standard application domain, required modifications of the paradigm underlying the concept of transactions. So far transactions were characterized through a short execution time and the preservation of the ACID properties. The introduction of engineering design transactions EDTAs led to different notions.

At the time being several different approaches are discussed in the literature (among others [BKK85], [KSUW85], [KLMP84]). The underlying system model, however, is basically the same for all these approaches [Kelter88]. A centralized data repository maintains the design data. A set of workstations, interconnected through a local area network, borrows design objects from the repository and transfers them to their local workspace. This is done by means of a CHECKOUT operation that sets a permanent lock on the copy of the design object in the repository. In his local DB the designer can perform a set of operations on the design object. These operations are either executed as conventional transactions or are CHECKOUT operations on object components. Objects are returned to the DBs they were borrowed from by means of a CHECKIN operation that releases the lock on the object.

Figure 3-5 shows the model of an engineering design system and the structure of an EDTA. A set of workstations is connected via a local area network. A database server, maintaining the central engineering design database forms the center of the network.

A user at workstation A requests an object obj_i from the central database (1), i.e. he/she checks it out there (2) and transfers it into his/her private local database (3). A user at workstation B then requests a part from obj_i from the user at workstation A. The procedure is the same as before. B requests a CHECKOUT from A (1), receives the desired data (2) and transfers it into the private local database (3). In the same way C checks out data from B. Thus we have an example for a three-level nested engineering design transaction as described in [BKK85], for instance.

When C has finished the processing of data received from B, the object is transferred back from the private local database of C to B by means of a CHECKIN operation (4). B can then return his part to A (4). Finally, A can checkin the processed object obj_i into the central database (4).

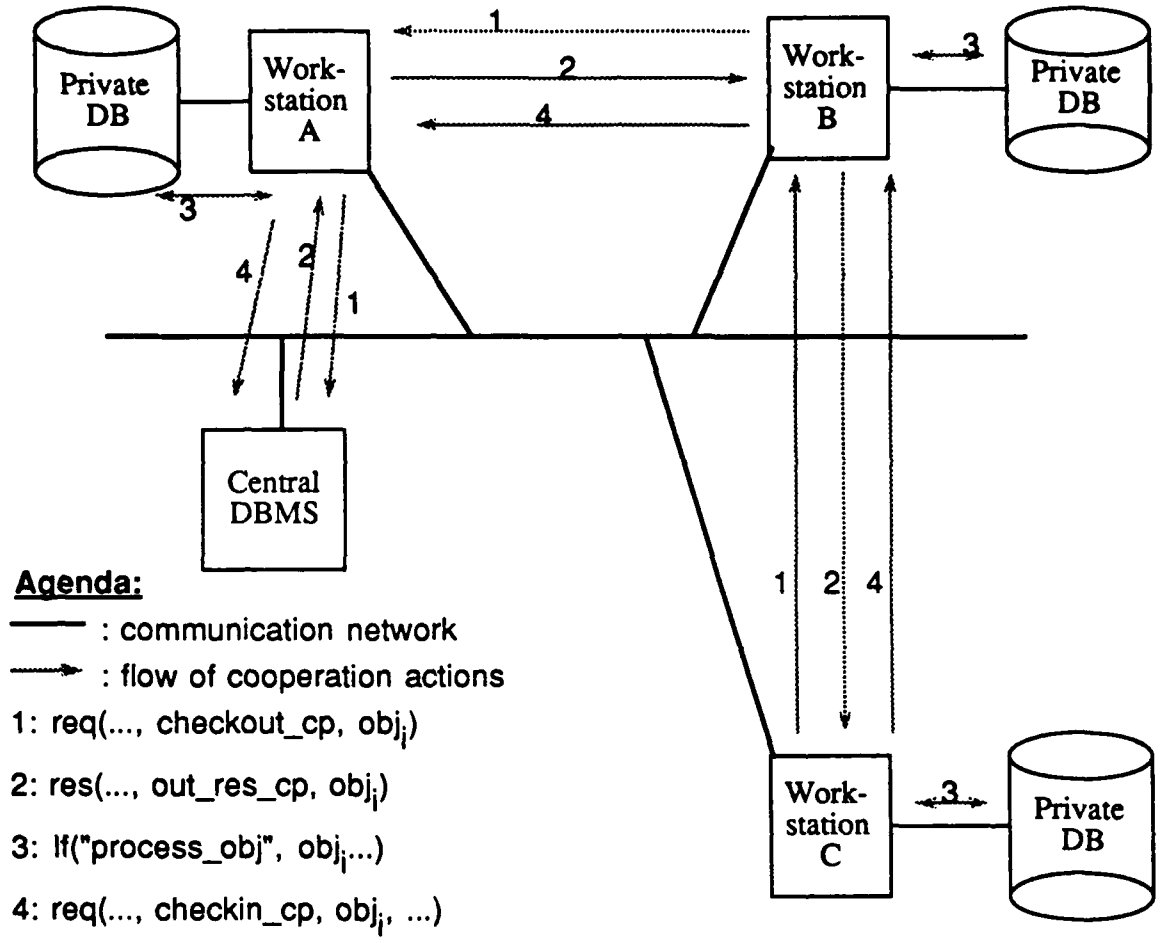

Figure 3-5: S-transactions for an engineering design environment

S-Transaction Template

The ST model of an EDTA envisages the existence of a CHECKOUT/CHECKIN service that is provided by each site of the engineering design environment, by the central repository as well as by the workstation DBs. Thus there is no difference between checking data out from the central site or from other workstation DBs. CHECKOUT and CHECKIN are represented through different continuation points. Each cp includes a local transaction by which the design object is copied from or into the database, respectively. As input parameters CHECKOUT LTs require a user identification, the identification of the design object and the checkout lock mode (e.g. read or modify). CHECKIN LTs have the design object itself and the user identification as parameters.

In the simplest case a designer transfers an object from a remote DB into his own, manipulates the object and returns the modified object. The corresponding ST requests a CHECKOUT service from the remote DB. Upon receipt of the desired design object, the object is transferred into the local DB by means of a corresponding LT. Having finished the

local processing, the design object is returned as a result to the CHECKIN service of the site, the object has been fetched from. This point needs a more detailed discussion.

If the object is simply transferred into the local DB by means of an LT we need a communication mechanism between the local DB and the ST in order to notify the ST of the termination of the local processing. In the simplest case the communication is implemented by a shared variable that is periodically polled by a continuation point of the ST. In a more sophisticated solution the cp is suspended until a signal from the local DB is received.

The first solution sketched can easily be implemented by means of currently available features of STDL. However, it lacks efficiency. The second, more efficient solution requires an extension of the current concepts that can be implemented easily. We introduce a new type of local transaction LTsig, that implies the suspension of the LT until a signal is received from the local site. The signal is specified as an argument of the LTsig call.

Both solution outlined so far are based on the assumption that the operations on the design object are performed under the control of the local site and are thus hidden from the ST Management. That means that in this case the S-transactions are only used for controlling the proper exchange of data between sites in accordance with a previously determined protocol. Although STDL provides all features of a universal programming language, it does not provide specific support for the manipulation of complex design processes. Hence, a specification of design operation in STDL is possible but not advisable.

3.5 SAGAs

While the transaction concepts discussed so far preserve the ACID properties of a transaction, SAGAs [GaSa87] deviate from that by disregarding isolation and thus providing what is called semantic atomicity, i.e. recovery of SAGAs is based on compensation. A SAGA consists of a set of conventional "flat" transactions and a corresponding set of compensating transactions that reverse the effects of the previously executed transactions. The component transactions of a SAGA commit prior to the end of the SAGA. Thus data that have been modified by a component transaction become accessible to other transactions outside the SAGA prior to the SAGA's end. As a consequence a (previously consistent) system state can only be restored if the data have not been accessed by other transactions before the failure occurred.

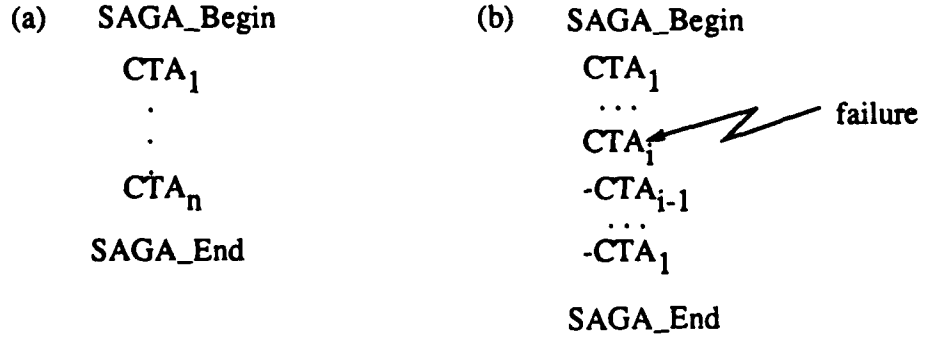

Figure 3-6: SAGA (a) and compensated SAGA (b)

By definition SAGAs represent a subset of S-transactions: a SAGA consists only of a set of local transactions. Nesting of SAGAs is not allowed. Hence they can be directly modeled by means of current STDL features.

For a set of LTs that are performed on a local site we retrieve the LT types, their parameters and their execution sequence from the log that is maintained by each STM. We can then call the compensating LTs in reverse order.

If a SAGA has involved multiple sites we get this information also from the STM's log and can initiate the corresponding compensations at the remote sites.

4.0 ADVANCED S-TRANSACTION STRUCTURES AND FEATURES

In the preceding section we have demonstrated how S-transactions can be used for implementing existing transaction concepts by using a uniform description language that has been made operational by a corresponding system architecture. In this section we will now outline some additional features of S-transactions and STDL that can be used for advanced applications.

4.1 Commit Protocols

In section 3.2 we have shown that the definition of a 2-phase commit protocol in STDL is a straightforward matter. A basic feature of STDL is the ability of defining communication protocols on a very high level of abstraction. Commit protocols, of course, are high level communication protocols. Thus STDL fits quite naturally as we outline in the following.

Regarding the DTA example, described in STDL in section 3.2, and comparing it with the state diagram for the 2-phase commit protocol, as described in [CePe84], we can recognize the following correspondences immediately.

The roles of coordinator and participant are reflected by the initiating classes the ST continuation points are assigned to.

The initial state of the coordinator corresponds to the combination of 'init_cp' and 'result_cp' (as in STDL results cannot be send to the requesting CP). The 'uncertain' state is represented through the suspension of the ST after having send the 'Prepare' message and while waiting for the 'Ready' or 'Abort' messages from the participants. Thus we better call it wait state. State transitions to 'abort' or 'commit' are equivalent to activating the continuation points 'ready_cp' or 'abort_cp' upon arrival of corresponding messages from the participants.

A participant, having executed the requested Sub_TA and returned the result, also enters a wait state, awaiting the 'Prepare' message from the coordinator. The activation of a participant's 'prepare_cp' upon arrival of the 'Prepare' message and the subsequent transmission of the 'Ready' message reflects the transition into the 'ready' state. The receipt of the coordinator's 'Commit' or 'Abort' message triggers the corresponding continuation point, representing the according state transition.

As a summary of the above comparison, we can state that continuation points represent states (except the 'wait' state that corresponds to the suspension of the ST execution), messages correspond to request or result messages in STDL.

Regarding now the state diagrams for the 3-phase commit protocol, it is obvious that it can be described in STDL easily, using the above sketched approach. The same strategy can also be applied to quorum-based protocols. The latter is even further supported by STDL as it is possible to define minima or maxima for messages to be received before activating a continuation point. As a consequence, we can specify the commit quorum or abort quorum for the corresponding continuation points, thus enabling the coordinator to react as soon as a decision has been achieved.

4.2 Non-Linear Transaction Structures

The transaction structures regarded so far were either linear or trees. Advanced distributed applications, however, might not be limited to these structures. STs are not limited to that either as we outline in the following. To some extent we can also model acyclic and even cyclic structures.

"Triangles" or cycles can easily be constructed in the following way. The initiating site A of an ST requests a service from a remote site B that includes the return of a result to A. If B, for some reason, is not able to provide the service, it may pass on the request to a site C. C computes the result and instead of returning it to B, C directly passes it to A. This is possible if C knows the ST identification at site A.

The name of the result receiving CP is known to C as C has to know the entire S-transaction description in order to be able to become involved in an ST. The identification of the receiving site must be passed from B to C. The transfer of a result to a site that is not yet involved in an ST is not allowed. The global ST identification must be known to a site before it is able to receive a result. Otherwise, it is possible that a site receives a result prior to the corresponding request. In that case a site cannot distinguish addressing errors from communication failures. Hence, an incidentally activated continuation point would run forever and could not be aborted.

In short, one kind of cyclic structures, as described above, are those that can be represented through a chain of requests with a final result transfer from the chain's end to its beginning. It is, however, also possible to perform cyclic requests, as the next example shows. A potential application for such cyclic requests could be the distributed evaluation of recursive operations, for instance.

Let A send a request to B. That means within the currently executed path of A's CP a request to another CP is coded. This CP path is supposed to be executed by B. In the trivial case, B returns a result to one of the CPs to be executed at A. This is the typical CALL - RETURN cycle, known from procedural programming languages. B could also request another service from A in order to be able to provide the service previously requested from A. For doing this there are two possibilities: either B can initiate another ST at A and transfer the result into the ST B actually executes or B can trigger the activation of a CP of the same ST.

Regarding these solutions isolated from other transaction structures, the first approach does not cause any problems while the second one does. The second approach implies the handling of multiple instances of the same continuation point within the same S-transaction that have different parent transactions. Thus, the management of aborts in general and partial roll-backs in particular become a problem.

Taking also other transaction structures into account like acyclic graph structures, specifically collapsed trees, even the first solution is not that easy to implement.

We can easily give examples for applications that require the synchronization of two parallel executing (sub) STs in one place (c.f transnational funds transfer in [Holtkamp88]). In those examples the thread of control forks by initiating CPs of the same ST type at remote sites. The CPs may be executed in parallel by the different sites. At some point of time these threads may synchronize, i.e. join at another site. How is it possible now to distinguish between joining requests and requests that are supposed to initiate different STs? In our current implementation we support only joins and no cyclic requests. All (subordinate) STs, belonging to the same global ST, carry the same global ST identification. If a request arrives at a site where the global ST identification is already known, this request is considered to be a joining one.

If the initiation of subordinate transaction is requested at a site that is already involved in a global transaction, the just described solution is no longer applicable. Instead, a construct is requested that allows to distinguish between joins and the initiation of another, subordinate transaction. In MUSE, the latter can be achieved by introducing a "REQ_SUB_ST" operation, for instance, that explicitely initiates a subordinate S-transaction.

4.3 Delegation of Functions and Transfer of Control

Another advanced feature of S-transactions is closely related to the issues discussed previously in this section. It is the possibility to delegate functions. A prerequisite for that is the ability to formulate transactions with non-linear structures. The following example aims at illustrating the delegation of functions (horizontal migration).

Assume that a site has initiated some kind of distributed transaction. At the very end of the transaction it is terminated by using a commit protocol. Normally, the initiating site also acts as the coordinator for the commit processing. The ability of sending results, for instance, to another than the requesting site enables the initiator of the entire transaction to transfer the commit processing to another site. As soon as the initiator has received the results from the subordinate remote sites he asks another site to play the role of a commit-coordinator. If the site agrees, the initiator sends 'Prepare' messages to the subordinate sites, indicating the newly found commit coordinator as the receiver of the 'Ready' messages issued by the subordinate sites.

The benefits of this possibility are manifold. For instance, the initiator is freed from some routine work, i.e. it is off-loaded. A careful choice of the commit coordinator in the above example can result in a significant decrease of communication costs. The appropriate selection of the commit coordinator can also result in a considerable performance enhancement.

Referring to the transnational banking scenario assume that the initiator of a transaction is located in the United States, let's say California, and the set of service providing sites

are located in Western Europe. If a European sites takes the role of the commit coordinator, communication distances are drastically reduced and thus most likely also communication costs. At the same time the difference in time zones falls away, reducing the problem of different business hours thus probably resulting in increased throughput.

A side effect of the delegation of functions is the restructuring of a transaction. The initiator of a transaction that is considered as the root node, can become a subordinate transaction in the above described example. If, for instance, the commit coordinator informs the initiator of the decision when having received the 'Ready' messages, the initiator can terminate the transaction while the commit coordinator still processes the acknowledgments of the subordinate sites.

In the above example the initiator still keeps control over the major part of the transaction, it has initiated, and transfers control only at the end for some routine task. We can easily imagine that S-transactions are not limited to that. It is also possible to move control at any time during the processing of a complex task. Let's regard another example from engineering design. According to the 'Waterfall' model for software development development phases are closed units of work that are sequentially ordered. If we describe this model in terms of S-transactions the transition from one phase to another corresponds to the transfer of control from one instance to another.

5.0 CONCLUSION

In this paper we have presented the concept of S-transactions and the corresponding definition language STDL as a flexible mechanism for describing the cooperation between nodes in a distributed system. The concept's flexibility is demonstrated by modeling different cooperation concepts in terms of S-transactions and STDL. The spectrum covered ranges from conventional transactions in a centralized database system over various types of distributed transaction concepts to mechanisms that provide for a higher degree of autonomy (e.g. SAGAs).

The practicability of the S-transaction concept and STDL have been proved by applying it successfully to application domains like transnational banking [Holtkamp88] or decentralized software development [Holtkamp89].

Our current and future work is related to fine tuning of STDL and of the underlying MUSE system architecture. We are also working on a rapid prototyping environment for the development of cooperating systems with MUSE and STDL as basis.

ACKNOWLEDGMENTS

I would like to thank my colleagues in Dortmund, specifically Dirk Ellinghaus, and Bernd Kraemer from the Naval Postgraduate School for their discussions and comments on earlier versions of this paper that heavily contributed to improve it.

REFERENCES

[BKK85] Bancilhon, F., Kim, W., Korth, H. F.
A Model of CAD Transactions
Proc. 11th Conference on Very Large Data Bases, Stockholm, 1985

[Becker89] Becker, D.
Remote Database Access in Computer Networks: Development of an ISO/RDA Component and Investigation of Integrating it into the MUSE Multidatabase System
Diploma Thesis, University of Dortmund, Computer Science Department, July, 1989 (in german)

[CePe84] Ceri, S. and Pelagatti, G.
Distributed Databases, Principles and Systems
McGraw-Hill, 1984

[ElVe88] Eliassen, F., Veijalainen, J. and Tirri, H.
Aspects of Transaction Modeling for Interoperable Information Systems
in: Speth, R. (ed.), Research into Networks and Distributed Applications, Elsevier Science Publishers B.V., North-Holland, 1988

[GaKo88] Garcia-Molina, H. and B. Kogan
Node Autonomy in Distributed Systems
Int. Symposium on Databases in Parallel and Distributed Systems, Austin, Texas, 1988

[GaSa87] Garcia-Molina, H. and Ken Salem
SAGAs
Proc. SIGMOD Conference, San Francisco, May 1987

[Garcia83] Garcia-Molina, H.
Using Semantic Knowledge for Transaction Processing in a Distributed Database
ACM Transactions on Database Systems, vol. 8, no. 2, 1983

[Gray78] Gray, J.
Notes on Database Operating Systems
in: LNCS vol. 60, Springer, 1978

[Gray81] Gray, J.
The Transaction Concept: Virtues and Limitations
Proc. 7th Int. Conference on Very Large Data Bases, Cannes, 1981

[Gray86] Gray, J.
An Approach to Decentralized Computer Systems
IEEE Transactions on Software Engineering, vol. SE-12, no. 6, June 1986, pp 684-692

[HaHo87] Hallmann, M. and B. Holtkamp
STDL: A Definition Language for Semantic Transactions
Proc. GI-Conference "Databases for Software Engineering", Dortmund, Nov. 87

[Holtkamp88] Holtkamp, B.
Preserving Autonomy in a Heterogeneous Multidatabase System
Proc. 12th International Computer Software & Applications Conference COMPSAC'88, Chicago, Oct. 1988

[Holtkamp89] Holtkamp, B.
MUSE - A Framework for Decentralized Software Development Environments
Technical Report No. 40, University of Dortmund, Department of Computer Science, Software Technology, August 1989

[ISO88] International Standardization Organization
Open Systems Interconnection - Distributed Transaction Processing Part 1: Model
ISO Draft Proposal 10026-1, 1988

[Johannson89] Johannsen, W.
Transactions in Federated Distributed Databases
Ph. D. Thesis, University of Frankfurt, Department of Computer Science, 1989

[KLMP84] Kim, W., Lorie, R., McNabb, D. and Plouffe, W.
A Transaction Mechanism for Engineering Design Databases
Proc. 10th Conference on Very Large Data Bases, Singapore, 1984

[KSUW85] Klahold, P., Schlageter, G., Unland, R. and Wilkes, W.
A Transaction Mechanism Supporting Complex Applications in Integrated Information Systems
Proc. ACM Int. Conference on Management of Data SIGMOD, 1985

[Kelter88] Kelter, U.
Concurrency Control and Recovery in Non-Standard Database Systems
Information Systems, vol. 13, 1988 (in german)

[KiLo89] Kim, W. and Lochowsky, F. H.
Object-Oriented Concepts, Databases, and Applications
Addison-Wesley, 1989

[MAP88a] SWIFT and University of Dortmund
S-Transactions
MAP Project 761B 'Multidatabase Services on ISO/OSI Networks for Transnational Accounting', Deliverable No. 6, University of Dortmund (eds.), Dec. 1988

[MAP88b] GMD-FOKUS, INRIA, SWIFT, University of Dortmund
Multidatabase System Architecture
MAP Project 761B 'Multidatabase Services on ISO/OSI Networks for Transnational Accounting', Deliverable No. 7, University of Dortmund (eds.), Dec. 1988

[StRo86] Stonebraker, M. and Rowe, L.
The Design of POSTGRES
Proc. ACM Conference on Management of Data SIGMOD, Washington D.C., May 1986

[Traiger83] Traiger, I.L.
Trends in Systems Aspects of Database Management
Proc. 2nd Int. Conference on Databases (ICOD-2), Wiley&Sons, 1983

[Wshop88] ACM SIGOPS European Workshop
Autonomy or Interdependence in Distributed Systems
Wolfson College, Cambridge (UK), Sept. 1988

Distribution List

SPAWAR-3242 Attn: Phil Andrews Washington, DC 20363-5100	1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 4401 Ford Ave. Alexandria, VA 22302-0268	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
John Maynard Code 402 Command and Control Departments Naval Ocean Systems Center San Diego, CA 92152	1
Dr. Sherman Gee ONT-221 Chief of Naval Research 800 N. Quincy Street Arlington, VA 22217-5000	1
Leah Wong Code 443 Command and Control Departments Naval Ocean Systems Center San Diego, CA 92152	1

Bernhard Holtkamp
Code 52Hk
Naval Postgraduate School
Monterey, CA 93943 — 5

Vincent Y. Lum
Code 52Lu
Naval Postgraduate School
Monterey, CA 93943 — 50

Neil C. Rowe
Code 52Rp
Naval Postgraduate School
Monterey, CA 93943 — 1

Klaus Meyer-Wegener
University of Kaiserslautern
Computer Science Department
P.O. Box 30 49
D-6750 Kaiserslautern
West Germany — 1