An Optimistic Transaction Model for a Disconnected Integration Architecture

Tim Lessner*, Fritz Laux†, Thomas Connolly*, Cherif Branki*, Malcolm Crowe*, and Martti Laiho‡
*School of Computing, University of the West of Scotland, name.surname@uws.ac.uk
†Fakultät Informatik, Reutlingen University, Germany, fritz.laux@reutlingen-university.de
‡ Dpt. of Business Information Technology, Haaga-Helia University of Applied Sciences, Finland, martti.laiho@haaga-helia.fi

***Abstract*—This work presents a disconnected transaction model able to cope with the increased complexity of long-living, hierarchically structured, and disconnected transactions. We combine an Open and Closed Nested Transaction Model with Optimistic Concurrency Control and interrelate flat transactions with the aforementioned complex nature. Despite temporary inconsistencies during a transaction's execution our model ensures consistency.**

***Index Terms*—Disconnected Transaction Management; Optimistic Concurrency Control; Advanced Transaction Models**

I. Introduction

Nowadays, Transaction Management (TM) must not only deal with short lived and flat transactions, TM must provide a transactional execution of long running and hierarchically structured business processes, so called complex transactions, involving many distributed loosely coupled, heterogeneous, and autonomous systems, represented as services as in Service Oriented Computing (SOC) for instance. To facilitate the loose coupling and increase the autonomy data should be modified in a disconnected and not in an online manner. By this we mean that the set of proposed modifications to data is prepared offline that requires a transacted sequence of operations on separate database connections. Further, message exchange, which takes place in such an architecture, is asynchronous and so is the data access, too. From a transactional view, the execution of a business process is a tree of interdependent and interleaved transactions, and during its execution the ACID (Atomicity, Consistency, Isolation, Durability) [1] properties are often weakened, and they allow for temporary inconsistencies to increase the performance. A locking isolation protocol, for instance, where other concurrent transactions read only committed results, leads to long blocking time caused by the process's duration and the asynchronous message exchange typical for disconnected architectures.

A disconnected nature overcomes this challenge for the price of a weakened isolation. The drawback of a weakened isolation is that other transactions can read pending results, which increases the danger for data to become inconsistent, and a widely used solution is compensation to semantically undo effects, also cascading effects, even after the transaction technically commits. Compensation can be interpreted as a necessity to conform to reality.

There are a couple of scenarios where a transaction becomes complex. The booking of a journey, for example, involves several heterogeneous web applications. Also cloud computing, where flat transactions access a highly replicated and distributed data storage, is confronted with a complex transaction structure.

Before the paper introduces the Transaction Model in section III, the problem is described by the next section, as well as the contribution of the paper. Related work is discussed in section IV and the Conclusion section completes this paper.

II. Problem Description & Contribution

One challenge is to provide a consistent transactional integration of heterogeneous systems, applications and databases across database (DB), middleware (MW) and application layer.

A lot of work has been carried out regarding (i) the transactional integration of heterogeneous systems (applications or databases) by the research community and industry (see [2][3][4]). In the relatively new domain of SOC many Web Service specifications cover a plurality of aspects and they all have in common that they provide a transactional interaction to a certain degree between services, or more generally components. These models have been motivated by the need for Business (BT) or User Transactions (UT). Moreover, (ii) transaction processing in DBMS has been addressed by a vast amount of research which focuses on the correctness, hence the serializability, of concurrent data processing in a transparent way by considering write and read operations only.

However, we believe the problem is that a common formal model integrating both aspects (i,ii) is missing, especially a model that considers the flat transactions as implemented by components and the overlying complex structure of transactions which guarantees consistency even if isolation or atomicity is weakened.

The contribution of this paper is a formal transaction model that interrelates the flat transactions as implemented by components with their composition and the resulting complex transaction structure. We impose an Optimistic Concurrency Control (OCC) structure within a component and require an OCC at the MW layer. The approach allows the detection of inconsistencies and furthermore decouples the concurrency control (CC) mechanism of the DB layer and thereby provides a solution for the "Impedance Mismatch" between (i) and

 ISBN:978-1-61208-115-1

(ii). We also believe that such a more integrated model helps to determine a trade-off between consistency, availability, and failure tolerance –user expectations– of a transactional integration system.

III. Transaction Model

We define a transaction t as a composition of several flat and short living transactions $t_1, t_2, \ldots, t_n$ where components $COMP = comp_1, \ldots, comp_j$ implement these transactions t_n. As in some MW specifications like the Java Enterprise Edition (JEE), components become part of the transactions, and they are in a transaction scope and bound to the life cycle of the component itself. In a disconnected architecture, however, these components lose their context in terms of the transaction as soon as the data is delivered. Components communicate with a data access layer asynchronously and no locks are kept because of the long-living and disconnected nature.

The overview in figure 2 shows the dependencies between the different concepts defined in this section.

A. Disconnected Transaction

A disconnected transaction has a read sphere sph^r and a write sphere sph^w. A sphere $sph \in SPH$ is defined as a set $sph = \{t_1, \ldots, t_n\}$ of transactions which logically groups transactions that belong together. Here, sph is a set of flat, short living, ACID transactions, and let transaction t be a sequence of operations $t = (op_1, \ldots, op_m)$ with $OP = \{op_1, \ldots, op_m\}$ as the set of data operations where each op_m is either of type $read(DO_m)$ or $write(DO_m)$, and let DO be the set of all data objects $DO = \cup_{k \in K} DO_k, K = \{1, 2, \ldots, m\}$. If there are versions of DO_k we denote a version DO_k^v, with superscript.

The disconnected behaviour is defined by $sph^r \cap sph^w = \emptyset$ and some transaction t is either in sph^r or sph^w.

Regarding the definitions introduced so far from an implementation point of view an implementation of dt, hence the several flat transactions in dt's write and read sphere, is required to initiate read and write transactions. Therefore, we define $comp$ as a component that implements dt. Additionally, we define a component to be in one of the following phases: reading (p1), disconnected and working (p2), validating (p3), and writing (p4). Hence we define a component to be in the phases similar to the phases of OCC. The difference, however, is the explicit disconnected and working phase. Notice, we impose this structure on a $comp$ which may be seen as a design guideline for the implementation of a single component.

Reading can be described as loading all the data required. After the modifications take place validation starts. Validation must be interpreted as a pre-phase of writing and only transactions $T^w \subseteq sph^w$ are allowed to enter p3 and p4 (transactions $T^r \subseteq sph^r$ can only enter the read phase). By following this structure a $comp$ can be seen as a component that follows the OCC paradigm introduced by [5].

Within an implementation of $comp$, write transactions may depend on each other and an explicit execution order like $t_1 \rightarrow t_2 \rightarrow t_4$ can be given. We define read transactions to not depend on each other because they can be parallelised without conflicting with each other.

Therefore, we define an explicit ordering over all writing transactions T^w and let Gdt be a directed, acyclic graph $Gdt = (Vdt, Edt)$ as defined in definition 1. The graph defines an execution order between two transactions $t_n \rightarrow t_o$ and Gdt is a partial order over the set of transactions $T'^w \subseteq (T^w \in sph^w)$. The set of transactions T''^w represents free transactions of a component $comp$, i.e., $T''^w \notin Vdt$. Hence, $T^w = T'^w \cup T''^w$.

The SAGA [6] model introduced the notion of compensation to semantically undo the effects of transactions. Compensation has been introduced to cope with the requirement for a weak isolation that arises if several sub-transactions form a long living process but each of the sub-transactions is allowed to commit and other transactions may read pending results. In case the transaction aborts its sub-transactions, whether committed or not, need to be undone, which is only possible by executing a compensation, e.g., to cancel a flight is the compensation of booking a flight. Our model also foresees compensation transactions to semantically undo the effects of a component, i.e., dt see III-C.

In our model a $comp$ either provides its own sequence of compensation transactions $comp_i^{-1} = (t_1, \ldots, t_n)$ or points to another component $comp_i^{-1} = comp_j$ representing the compensation.

Finally, we define a disconnected transaction as:

Definition 1: Disconnected transaction dt

(1) A Disconnected transaction is defined as $dt := (sph^r, sph^w, Gdt, comp, comp^{-1})$ with read sphere $sph^r = (t_1, \ldots, t_n)$, write sphere $sph^w = (t_1, \ldots, t_m)$, a partial order $Gdt = (Vdt, Edt)$ with $Vdt \equiv T'^w \subseteq T^w$ and the set of edges Edt is defined as $\forall t_n, t_o \in V : t_n \rightarrow t_o \Leftrightarrow edt \in Edt$ with $edt = (t_n, t_o)$. Let $comp$ be the component that implements dt, and let $comp^{-1}$ be the compensation handler of dt. And, $\forall comp \in COMP$: $comp$ is in exactly one phase p1,p2,p3 or p4.

(2) $\forall dt \in DT : sph^r \cap sph^w = \emptyset$

(3) dt only commits successfully if all T^r and T^w commit successfully.

(4) In the case of failure the $comp^-1$ must be executed to semantically undo the changes of dt.

B. Consistency of dt

Due to the long-living and disconnected nature it would be not favorable to lock data for as long as dt lasts because locking hinders global progress of the transaction. However, we must restrict this statement. No locking refers to the entire life-cycle (p1-p4) of dt, and locking for the short living transactions T is allowed, because they release their locks with their commit and so, the blocking time is reduced to the time validation and writing takes place (p3,p4). An exclusive access during the validation and writing phase is necessary because a consistent snapshot of a data object that has to be validated is required and modifications must be written back eventually.

 ISBN:978-1-61208-115-1

We aim for a more decoupled transaction management. This in turn requires validation to prevent from read and write anomalies, like lost update. Our model foresees validation to take place at the MW layer. By applying validation at the MW layer we can basically decouple the DBMS in a sense that consistency is already provided by the MW. To ensure consistency at the MW the validation mechanism must perform a validation as introduced by Kung and Robinson[5].

Definition 2: Validation

(1) Let $RS(t_i)$ be the set of DO_i that is read by t_i of $comp$ and let $WS(t_n)$ be the set of DO_m eventually modified by t_n of $comp$. Also, let $ts(RS(t_n))$ be the timestamp of the read set of t_n. Only if $ts(RS(t_n)) < ts(RS(t_o)) \wedge RS(t_n) \cap RS(t_o) = \emptyset \wedge WS(t_n) \cap WS(t_o) = \emptyset \wedge WS(t_n) \subseteq RS(t_n)$ holds, validation $val(DO_k^v) \rightarrow DO_k^{v+1}$ is conflict free. Let val be an algorithm that detects conflicts between two versions DO_k^v and DO_k^{v+1} of a data object by applying these aforementioned rules.

(2) Further we require the validation to be "escrow serializable" [7] and to obey the order defined by Gdt.

By applying this validation schema outdated data, i.e., data that has been modified by other transactions during the execution of dt, can be detected and anomalies can be prevented. In our previous work [7] about optimistic validation in disconnected and mobile computing we introduced "escrow serializability" *ec* and a "reconciliation mechanism" that allows the number of validation conflicts to be reduced by automatically replaying a certain class of operations. We require the validation to be *ec*, and the execution of each t is correct, if it is *ec* serializable.

Another issue which has to be considered is the possible existence of so called atomic units within sph^w. Atomic units exist if some transactions T^w are commit or abort dependent to each other, i.e., transaction t_n is only allowed to commit if transaction t_m does. To depict such dependencies we have to extend our model.

1) Atomic Units: An atomic unit groups several flat transactions where an atomic outcome of the group is required. The members of an atomic unit become sub-transactions. Considering the concept of atomicity the concept of an atomic unit is ambiguous because something that is atomic now consists of other atomic units, namely each sub-transaction itself. The point is that atomic refers to the expected outcome and the execution of several sub-transactions must be atomic. In other words, atomicity is relative to the level of the transaction tree.

We introduce the notion of an atomic unit $au = (t_1, \ldots, t_n)$ as a sequence of flat transactions, and let $AU_i = \{au_{i,1}, \ldots, au_{i,h}\}$ be the set of all atomic units of dt_i which form an imposed structure on sph_i^w; thus $sph_i^w := AU_i$.

Definition 3: Atomic unit au of dt

(1) Let $AU_i = \{au_{i,1}, \ldots, au_{i,h}\}$ be the set of all atomic units of dt_i which is an imposed structure on sph_i^w; thus $sph_i^w := AU_i$. Further, let $au_{i,h} = (t_1, \ldots, t_n)$ group $(t_1^w, \ldots, t_n^w) \in T^w$ into an indivisible group of transactions where either all $T^w \in au_{i,h}$ commit or abort. And, let AU be the set of all AU_i

Now, let $csDO_i$ be the change set –modifications– of data objects modified by dt_i and $csDO_i(au_{i,h})$ the change set of $au_{i,h}$. The validation must now ensure that only if each $csDO_i(au_{i,h})$ passes validation the modifications are written back to the database. To achieve an atomic outcome for sph_i^w we need a Closed Nested Transaction CNT [8], [9] structure that is able to guarantee an atomic outcome of sph_i^w. We need a CNT because of the imposed order and the atomic units which may encompass several t.

Definition 4: Closed Nested Transaction CNT

(1) Let $CNT(sph_i^w)$ be a closed nested transaction over sph_i^w which is defined as a tree $CNT(sph_i^w) = (Vcnt, Ecnt)$ with sph_i^w as root node r which only commits if all its children commit, the set of vertexes $Vcnt \equiv AU_i$ and the set of edges $Ecnt$ is defined as $\forall au_{i,h}, au_{i,j} \in Vcnt : au_{i,h} \rightarrow au_{i,j} \Leftrightarrow ecnt \in Ecnt$ with $ecnt = (au_{i,h}, au_{i,j})$.

(2) If there exists an order (edge) $\exists edt \in Edt = t_m \rightarrow t_n$ with $t_m \in au_{i,h}$ and $t_n \in au_{i,j}$ for $h \neq j$ then there must be a dependency between $au_{i,h}$ and $au_{i,j}$ so that $t_m \rightarrow t_{first} \in au_{i,j}$, and $t_{last} \in au_{i,j} \rightarrow t_{m+1} \in au_{i,h}$, with t_{first} as the first and t_{last} as last element in $au_{i,j}$, and let t_{m+1} be the successor of t_m so that $t_m \rightarrow t_{m+1}$. Thus $au_{i,j}$ becomes part of $au_{i,h}$. If $\neg\exists(t_m \rightarrow t_{m+1})$ then $au_{i,j}$ runs concurrent to $au_{i,h}$.

(3) Further, the order of atomic units must terminate.

Example 1: CNT

Given $sph^w = (au_1, au_2, au_3, au_4, au_5)$ with $au_1 = (t_1, t_2, t_3, t_4)$, with $au_2 = (t_5, t_6)$, $au_3 = (t_7, t_8)$, $au_4 = (t_9)$, $au_5 = (t_{10})$, and

$$Gdt = (Vdt = \{t_1, t_3, t_4, t_5, t_6, t_7, t_8, t_9\}, Edt = \{(t_1, t_3)(t_3, t_6)(t_3, t_4)(t_5, t_6)(t_7, t_8)(t_7, t_9)\})$$

The resulting CNT, i.e., the execution model, is shown in Fig. 1.Notice, au's are shown as dashed lines. As shown, au_2 becomes part of au_1 because of the edges (t_3, t_6) and (t_3, t_4) defined in Gdt. Since there is no order defined between t_1, t_2 it is possible to parallelise t_2.

Regarding $au_3 = (t_7, t_8)$ and $au_4 = (t_9)$ the situation differs. Due to the order relations (t_7, t_8) and (t_7, t_9) au_3 and au_4 must be executed serial.

To achieve an atomic outcome of CNT a Two-Phase-Commit (2PC) is required between t_2 and the transactions t_3, t_5, t_6, t_4 with t_1 as coordinator. One possibility to achieve an atomic outcome of t_3, t_5, t_6, t_4 is to introduce a new atomic unit au_1'. We regard this issue as implementation detail and leave an answer open for future research. However, a coordination between each chain in a branch is required. For example, the coordinator initiates t_3, awaits the result, in case the result is a pre-commit, it initiates t_5, then t_6 and finally t_4 if all of them sent their pre-commit au_2 is committed and the result is sent back as a pre-commit to t_1. Between au_1, au_3, and au_5 a 2PC is required.

 ISBN:978-1-61208-115-1

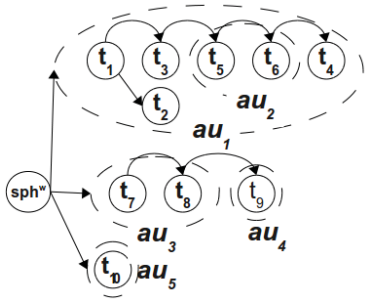

Fig. 1. Exemplary CNT (straight lines show possible parallelisation, curved lines show an order)

According to the example some further specification for CNT is required to ensure an atomic outcome:

Definition 5: Commitment rules of CNT

(1) A parent is only allowed to commit if all its children commit.
(2) Between all siblings of CNT a Two-Phase-Commit (2PC) is required.
(3) Between a chain of au in a branch an external coordinator must ensure:
(a) A successor of $au_{i,h}$ is only allowed to start if its predecessor $au_{i,h-1}$ pre-commits.
(b) If one $au_{i,h}$ in the chain aborts all other au have to abort, too.
(c) If all au sent their pre-commit to the coordinator the coordinator sends a commit message to all au and each au has to commit, i.e., a commit is a promise by the au
(d) A DO can be passed from a au only to its predecessor.

Notice the difference between a parallel execution which requires a 2PC and a chained one which requires coordination for an ordered execution. Both have in common that a commit is a promise that requires the pre-commit state and if only one of the involved parties aborts, the entire au must be aborted. Following the rules defined in Definition 5 compensation is not required, and recovery is possible because locks are released after a commit of all involved transactions (compensation will play an important role later in section III-C). Also, since a DO is only passed between already pre-committed, au isolation is maintained too. For further details in the domain of nested transactions we refer to the seminal work [8], [9], [10], [1], [11].

Now, we consider the concurrent execution of several dt. Usually a correct, i.e., consistent, concurrent, execution is given if the conflict graph between all transactions, dt in our case, is acyclic. Our model, however, does not require such a conflict graph and we rely on the optimistic validation instead.

The concurrent execution of several au of different dt is correct if OCC is able to detect conflicts between au that are either a sequence of transactions or single transactions, and if the order within and between atomic units is obeyed. Hence, an arbitrarily ordering of concurrent au is limited to the imposed order defined, but free transactions, i.e., atomic units, which are not part of the order relation can interleave with ordered atomic units in any way. Further, bear in mind that OCC is a first wins strategy.

A concurrent execution of several $AU = (AU_i = (au_{i,1}, \ldots, au_{i,j}), AU_k = (au_{k,1}, \ldots, au_{k,l}), \ldots)$ must be consistent because the OCC validation detects read-write or write-write conflicts for some data object DO (see Definition 2) and an arbitrary interleaving of free transactions with ordered au. In case of a conflict the transaction will be aborted. Further, the CNT ensures an atomic and non-isolated outcome. Hence, without providing a proof (see section V) OCC and CNT should ensure a consistent and atomic outcome of a concurrent execution of atomic units.

C. User Transaction ut

As already mentioned, a transactional integration has to deal with a complex hierarchically structured transaction, referred to as user transaction ut. An ut is a composition of several components that interact with each other in a defined order.

Definition 6: User transaction ut

Let $UT = ut_1, ut_2, \ldots, ut_m$ be the set of all user transactions. We define an $ut_m := (DT_m, Gut_m, AUT_m)$ with DT_m as the set of disconnected transactions composed by ut. Gut_m is a directed, acyclic graph with $Gut_m = (Vut_m, Eut_m)$ with $Vut_m \equiv DT_m = dt_{1,m}, \ldots, dt_{i,m}$ and the set of edges Eut_m is defined as $\forall dt_{m,i}, dt_{m,n} \in Vut_m : dt_{m,i} \rightarrow dt_{m,n} \Leftrightarrow eut \in Eut_m$ with $eut = (dt_{m,i}, dt_{m,n})$. The dependency $dt_{m,i} \rightarrow dt_{m,n}$ expresses an execution order, hence Gut_m represents a partial order over the set of disconnected transactions DT_m for ut_m. AUT is defined in the following section.

In the following we consider the existence of atomic units within an ut and how we can achieve an atomic outcome of ut.

D. Consistency of ut

As already investigated by the research community a complex transaction, like ut, must cope with sub-transactions defined by transactional boundaries within the execution model. A transactional boundary demarcates parts of the complete transaction and defines thereby sub- transactions, or in other words atomic units (atomic refers to the outcome). These units can now be interleaved in a concurrent execution which leads to an increased performance because not an entire ut must be scheduled.

Further, atomic units can be exploited to allow for a partial rollback of ut if the sequence of corresponding compensation steps is defined. This is also known as the "Spheres of Joint Compensation" model introduced by [12]. More details about the settings and the technological realisation of transactional boundaries can be found in [13], [14], [15].

We provide a more general notion of atomic units and focus on the interrelation between these more high level units and above defined disconnected transactions.

 ISBN:978-1-61208-115-1

1) Atomic Units of ut: The following definitions are analog to the ones in section III-A above.

Definition 7: Atomic unit aut of ut

Let $AUT_m = \{aut_{m,1}, \ldots, aut_{m,j}\}$ be the set of all atomic units of ut_m with $aut_{m,j} = (dt_1, \ldots, dt_i)$, and with $(dt_1, \ldots, dt_i) \in Eut_m$. Atomic unit $aut_{m,j}$ groups the spheres of disconnected transactions into an indivisible group where either all $dt_i \in aut_{m,j}$ commit or abort. Hence, AUT_m is an imposed structure on an ut. And, let AUT be the set of all AUT_i.

In contrast to CNT which ensures an atomic outcome after the validation takes place, a similar structure is required that coordinates the validation for several dt, i.e., their write spheres, grouped in an aut. Also, amongst all aut. Thus, a transaction structure for the validation is required. Further, the commitment rules and the coordination amongst the dt is different to CNT. This is caused by the time between the termination of two dt. Within one dt it is possible to bring in the modifications of one atomic unit of dt in one step (there is only one write phase), at this stage, one dt_i may enter its termination (p3,p4) a long time before another dt_k of the same atomic unit enters its termination.

Eventually, this means that isolation is weakened and compensation is required to semantically undo the effects because a dt has to commit to release its locks and isolation is no longer possible. This requires, as defined in definition 1, each dt to define its compensation, which may lead to a less favourable cascading compensation. We refer to [6], [12], [16] for a thorough discussion about compensation. Isolation is no longer given, but since each t and, so each dt, must pass the validation, our approach is able to detect and prevent inconsistencies albeit after they arise, which we nevertheless consider as a clear improvement because inconsistencies at the DB level can be avoided despite a weak isolation.

Now, the coordination mechanism on top of the validation is introduced. Similar to CNT we introduce an Open Nested Transaction ONT which is considered open because of the weak isolation.

Definition 8: Open Nested Transaction ONT

(1) Let $ONT(dt_i)$ be an open nested transaction over ut_i which is defined as a tree $ONT(ut_i) = (Vont, Eont)$ with ut_i as root node r which only commits if all its children commit, the set of vertexes $Vont \equiv AUT$ and the set of edges $Eont$ is defined as $\forall aut_{m,j}, aut_{m,k} \in Vont : aut_{m,j} \rightarrow aut_{m,k} \Leftrightarrow eont \in Eont$ with $eont = (aut_{m,j}, aut_{m,k})$.
(2) If there exists an order (edge) $\exists eut \in Eut_m = dt_i \rightarrow dt_l$ with $dt_i \in aut_{m,j}$ and $dt_l \in aut_{m,k}$ for $j \neq k$ then there must be a dependency between $aut_{m,j}$ and $aut_{m,k}$ so that $dt_i \rightarrow dt_{first} \in aut_{m,k}$, and $dt_{last} \in aut_{m,k} \rightarrow dt_{i+1} \in aut_{m,j}$, with dt_{first} as the first and dt_{last} as last element in $aut_{m,k}$, and let dt_{i+1} be the successor of dt_i so that $dt_i \rightarrow dt_{i+1}$. Thus $aut_{m,k}$ becomes part of $aut_{m,j}$. If $\neg\exists(dt_i \rightarrow dt_{i+1})$ then $aut_{m,j}$ runs concurrent to $aut_{m,k}$.
(3) Further, the construction of an order of atomic units must terminate.

Please bear in mind that a single dt is also encapsulated by an aut and that a dt commits if its wsp^w does. So, actually ONT is a structure over all wsp^w which are themselves structured by CNT. Whereas ONT coordinates above the validation, CNT coordinates below. To ensure an atomic outcome for a dt some further specification for ONT is required:

Definition 9: Commitment rules of ONT

(1) A parent is only allowed to commit if all its children commit. If one of its children abort compensation is required.
(2) Between a chain of aut the following holds:
(a) A successor of $aut_{m,j}$ is only allowed to start if its predecessor $aut_{m,j-1}$ commits.
(b) If one aut in the chain aborts all other aut have to abort too, and the compensation handler of each dt of the same aut must be called in reverse order as defined by Gut

The commitment rules of ONT are different from the ones defined for CNT due to the open nature. As a concrete realisation we propose to lodge an execution plan by a coordinator (CO) for each ut, hence the model of the ONT itself. Each dt which enters its write sphere has to register and must (i) await the CO's acknowledgement to enter the validation and (ii) it has to send the final outcome, commit or abort, to the CO. If CO receives a commit message from a dt it marks the corresponding node in the graph as committed, and in case of an abort CO does not only mark the node as aborted, it must initialise the compensation too. To control the compensation a compensation model $comp^{-1}(ut)$ must be derived. We omit a definition and the interested reader is referred to [12].

A dt is only allowed to enter the validation if all its predecessors have already committed. A parent of several siblings is only allowed to commit if all its children have committed. To release locks only if all siblings commit is not required because of the compensation, hence a 2PC [1] in its classic definition is not required. Following this approach the CO can obey the order and the status of each dt. The suggested approach can be interpreted as reducing a graph to its root node.

To complete the model we must show that the concurrent execution of atomic units aut is also consistent. The argumentation follows the same way as at the end of section III-B.

IV. Related Work

The Nested Transaction Model has been introduced by Moss [8] and can be seen as the seminal work concerning Advanced, Workflow, or Business Transaction Models (see [2], [4]). Spheres have been introduced by Davies [17] and can be seen as the generalisation of the Nested Transaction Model which itself can be seen as a generalisation of the chained transaction model [1].

Compensation was already foreseen by Moss, but especially the SAGA [6] model heavily applied compensation transactions. The work by Leymann [12] especially focused on the existence of atomic and compensation spheres. One influential workflow transaction model is Reuters Contract model [18]

 ISBN:978-1-61208-115-1

which is a conceptual framework for the reliable execution of long-lived computations in a distributed environment. OCC was introduced by Kung and Robinson [5] and even if it never gained a lot of attention as a CC mechanism in a DBMS, its validation concept has been applied in synchronisation concepts in Mobile Computing, for example, but rarely adopted into the MW itself. The PyrrhoDB [19] is the only database we are aware of that implements OCC as CC mechanism. Laiho and Laux [20] thoroughly analysed Row Version Verification (RVV) as an implementation of OCC within a disconnected architecture and their work provides, beside a detailed discussion, patterns to implement RVV for a couple of common databases and data access technologies at the MW layer. Fekete et al. [21] also pointed out that an integration of underlying short transactions and complex transactions is important, and mechanisms to ensure consistency without the need to lock data are required. Their work, however, introduces research directions and not a formal model describing the interdependence between flat and complex transactions.

V. Conclusion and Future Work

The defined transaction model decouples the DB from the MW layer by applying an OCC at the MW layer and imposes an OCC phase model within a component. This enables to detect overall inconsistencies despite temporal sub-transactional inconsistencies. We also interrelated flat transactions with the complex transaction structure by applying a CNT after the validation to ensure an atomic outcome, and ONT coordinates the atomic outcome of ut above the validation. This interrelation closes the gap between flat and complex transactions, and the application of OCC at the MW can also help to establish a loose transaction coupling between DB and MW.

Our future work will include a theoretical underpinning, especially proofs, and focus on an implementation combining our previous work [7] with this work. Our specific focus is thereby on the semantics of transactions, as well as how their transactional requirements can be expressed. On that account we are working on the quantification of consistency requirements. This abstract model will form the basis for our future work.

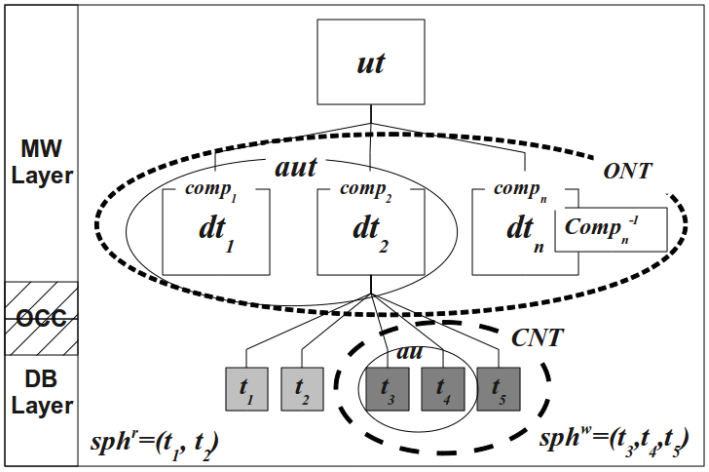

Fig. 2. Dependencies between ut, dt, $comp$, t, CNT and ONT

References

[1] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[2] Sushil Jajodia and Larry Kerschberg, editors. *Advanced Transaction Models and Architectures*. 1997.

[3] Ahmed Elmagarmid, Marek Rusinkiewicz, and Amit Sheth, editors. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[4] Ting Wang, Jochem Vonk, Benedikt Kratz, and Paul Grefen. A survey on the history of transaction management: from flat to grid transactions. *Distrib. Parallel Databases*, 23(3):235–270, 2008.

[5] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, 1981.

[6] Hector Garcia-Molina and Kenneth Salem. Sagas. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 249–259. ACM, 1987.

[7] Fritz Laux and Tim Lessner. Escrow Serializability and Reconciliation in Mobile Computing using Semantic Properties. *International Journal On Advances in Telecommunications*, 2(2):72–87, 2009.

[8] J. Eliot B. Moss. *Nested transactions: an approach to reliable distributed computing*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.

[9] J. Eliot B. Moss. Open nested transactions: Semantics and support. *Workshop on Memory Performance Issues*, 2006.

[10] Gerhard Weikum and Hans-Jörg Schek. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. Morgan Kaufmann, 1992.

[11] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.

[12] Frank Leymann. Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems. In *BTW*, pages 51–70, 1995.

[13] Michael Beisiegel et al. Service component architecture (sca) v1.00, 2010-11-08, 2007.

[14] Olaf Zimmermann, Jonas Grundler, Stefan Tai, and Frank Leymann. Architectural Decisions and Patterns for Transactional Workflows in SOA. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 81–93. Springer-Verlag, 2007.

[15] Heiko Schuldt, Gustavo Alonso, Catriel Beeri, and Hans-Jörg Schek. Atomicity and isolation for transactional processes. *ACM Trans. Database Syst.*, 27(1):63–116, 2002.

[16] Heiko Schuldt and Gustavo Alonso and Hans-Jörg Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 316–326. ACM Press, 1999.

[17] C. T. Davies. Data processing spheres of control. *IBM Systems Journal*, 17(2):179–198, 1978.

[18] Andreas Reuter and Kerstin Schneider and Friedemann Schwenkreis. ConTracts Revisited. In Sushil Jajodia and Larry Kerschberg, editors, *Advanced Transaction Models and Architectures*. 1997.

[19] Malcolm Crowe (University of the West of Scotland). The Pyrrho database management system (http://www.pyrrhodb.com/, 2010-11-02), 2010.

[20] Martti Laiho and Fritz Laux. Implementing optimistic concurrency control for persistence middleware using row version verification. In Fritz Laux and Lena Strömbäck, editors, *The Second International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2010)*, pages 45–50. IEEE Computer Society, 2010.

[21] Alan Fekete, Paul Greenfield, Dean Kuo, and Julian Jang. Transactions in loosely coupled distributed systems. In *Proceedings of the 14th Australasian database conference - Volume 17*, ADC '03, pages 7–12, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

 ISBN:978-1-61208-115-1