***I. J. Computer Network and Information Security*, 2019, 9, 24-39**
Published Online September 2019 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2019.09.04

An Approach to Develop a Transactional Calculus for Semi-Structured Database System

Rita Ganguly
Department of Computer Applications; Dr.B.C.Roy Engineering College; Durgapur: 713206; West Bengal, India
E-mail: ganguly.rita@gmail.com

Anirban Sarkar
Department of Computer Science; National Institute of Technology; Durgapur;713209; West Bengal, India
E-mail: sarkar.anirban@gmail.com

Received: 06 August 2019; Accepted: 25 August 2019; Published: 08 September 2019

Abstract—Traditional database system forces all data to adhere to an explicitly specified, rigid schema and most of the limitations of traditional database may be overcome by semi-structured database. Whereas a traditional transaction system guarantee that either all modifications are done or none of these i.e. the database must be atomic (either occurs all or occurs nothing) in nature. In this paper transaction is treating as a mapping from its environment to compensable programs and provides a transaction refinement calculus. The motivation of the Transactional Calculus for Semi Structured Database System (TCSS) is-finally, on a highly distributed network, it is desirable to provide some amount of fault tolerance. The paper proposes a mathematical framework for transactions where a transaction is treated as a mapping from its environment to compensable programs and also provides a transaction refinement calculus. It proposes to show that most of the semi structured transaction can be converted to a calculus based model which is simply consists of a forward activity and a compensation module of CAP (consistency, availability, and partition tolerance) [12] and BASE (basic availability, soft state and eventually consistent) [45] theorem. It proposes to show that most of the semi-structured transaction can be converted to a calculus based model which is simply consists of a forward activity and a compensation module of CAP and BASE theorem. It is important that the service still perform as expected if some nodes crash or communication links fail, Verification of several useful properties of the proposed TCSS includes in this article. Moreover, a detailed comparative analysis has been providing towards evaluation of the proposed TCSS.

Index Terms—Semi-structured, transactional calculus, X-Query, GOOSSDM, CAP, BASE, GQL-SS.

I. Introduction

In recent years, researches have produced several proposals [2, 3, 4, 5, 7, 8, and 9] towards conceptual modelling of semi-structured database system compare to the proposals of conceptual modelling. To overcome traditional transactional problems, extending the transactional processing system in semi-structured database by addition of compensation and coordination of consistency, availability, and partition tolerance (CAP)[12] and basic availability, soft state and eventually consistent (BASE) [45] theorem and enrich a standard design model with new healthiness conditions. There is no specific transactional calculus for semi-structured data. The proposed Transactional Calculus for Semi-structured database (TCSS) puts forward a mathematical framework for transactions where a transaction is treated as a mapping from its environment to compensable program. Further, the transactional calculus is derive from an algebra based query language GQL-SS [11] and illustrated using examples of real life. The motivation of the Transactional Calculus for Semi-structured System, it is desirable to provide some amount of fault tolerance, on a highly distributed network. It is important that the service still perform as expected, when some nodes crash or communication links fail. The ACID (Atomicity, Consistency, Isolation and Durability) acronym says that database transactions should be f**irstly**, seem indispensable, and yet they are incompatible with availability and performance in very large systems. The semi-structured database violates the ACID properties. According to ACID properties in Atomic the entire transaction will fail if one node element of a transaction fails, but in semi-structured database, it is not possible. In semi-structured database, if one node is damaged the entire network should not be affected. **Secondly**, no transaction has access to any other transaction in Isolation that is in an intermediate or unfinished state. Thus, each transaction is independent unto itself. This is required for both performance and consistency of transactions within a database. The semi-structured database violates this property because it works in path basis and every node is inter linked to each other. The benefits of the transactional calculus for Semi-structured databases are manifold. It provides supports towards (1) structural and functional design concerns with enriched semantics and syntaxes for transactional calculus of semi-structured database represented by precise knowledge of domain independent conceptualization;(2) a systematic methodology which used to transforming calculus for functional design; (3)Transactional Calculus to Semi-structured database query system provides guidelines for the purpose of mapping .The proposed Transactional system for semi-structured is based on path expression. The path expressions may also contain label variables to preserve labels or tags. Three types of algorithms are using to evaluate the path in Graph Object Oriented Semi-Structured Data Model (GOOSSDM)[2, 19, 20, and 21] schema and Graphical Query Language for Semi-structured (GQL-SS) [11] schema, one for searching return node, second for searching the path from root of GOOSSDM schema to the desired node and the third one is for the searching and listing of the tail nodes.. Here trying to use the CAP theorem in the broader context of distributed computing theory. An important contribution of this paper is to discuss some of the practical implication of CAP Theorem of a transactional calculus for Semi-structured database. There are some proposal; they are only using CAP [12] or BASE [25] theorem or without these. To introduce the transactional calculus for Semi-structured database, with the help of CAP theorem, the CAP theorem was introducing as a trade-off between consistency, availability and partition tolerance. **Consistency**: A read sees all previously completed writes i.e. all nodes see the same data at the same time. **Availability**: A guarantee that every request receives a response about whether it succeeded or failed i.e. read and write always succeed. This means that in GOOSSDM schema there should be a searching path and its return some value. The path value should not be null. **Partition Tolerance**: Guaranteed properties are maintained even when network failures prevent some machines from communicating with others. The system continues to operate despite arbitrary partitioning due to network failures.

However, developers face some challenges despite of several advantages of existing Semi-structured databases, when they apply the transaction processing system. Such challenges are as follows-

Ch1: Lack of transactional methodology that blends semi-structured databases specification with syntaxes of transactional calculus for semi-structured database system.
Ch.2: Majority of existing transactional procedure are not usable for large semi-structured database queries.
Ch.3: Few transactional calculus for semi-structured database approaches are present in literatures that may represent evolving knowledge of transaction in semi-structured databases but not in precise.
Ch.4: Appropriate guidelines and tools are absent which may help designers for specification.
Ch.5: XML-based semi-structured database systems characterized by an expressive global schema. The main issue here concerns the presence of a significant set of integrity constraints expressed over the schema and the concept of node identity, which requires particular attention when data come from autonomous data sources.

This paper fulfils the deficiency of systematic methodology in transactional calculus of GOOSSDM model[44]. The paper is structuring as follows. Several related works in this field specified in Section 2 briefly. Section 3 is about the GOOSSDM modelling framework and this portion is subdividing into two parts components of GOOSSDM and Illustration of GOOSSDM. The proposed Transaction calculus for semi-structured database system (TCSS) has been describing and formalised in Section 4. Next, guidelines about the way in which the validation of TCSS can be applied databases by using CAP and BASE theorem and application specific conceptualisations have been suggesting in Section 5. Further, the proposed TCSS have been implementing and visualised using different operators and practically illustrates the proposed work using suitable example in Section 6. Following this, Section 7 practically illustrates the proposed work using a suitable programming code. Finally, the paper is concluding in Section 8.Aiming to overcome issues explained in above mentioned challenges this paper proposes several objectives. First, the proposed framework of Transactional system for semi-structured is based on path expression. They may also contain path variables, which, are evaluating to the empty path or to a path having a length of n edges. The path expressions may also contain label variables to preserve labels or tags. At second, the path operator is using to set the root node in GOOSSDM [2, 19, 20, and 21] schema and useful to find the path from the root node to desired node for any transaction. At Third, the propose work facilitate the early verification of the semi-structured data schema structure in correspondence with the desired transactional calculus. Finally, the transactional calculus is introducing to Semi-structured database, with the help of CAP and BASE theorem. This objective addresses the issues described in Ch.2, Ch.3, Ch.4 and Ch.5.The benefits of the Transactional Calculus for Semi-structured system will represents a framework for specifying the semantics of a transactional facility integrated within a Semi-structured database system. The motivation of the Transactional Calculus for Semi-structured System is-finally, on a highly distributed network, is that when some nodes crash or communication links fail, it is important that the service still perform as expected. This paper fulfils the deficiency of systematic methodology in transactional calculus of GOOSSDM model. In addition, this paper proposes a formal transactional calculus called

Transactional Calculus for Semi-structured database (TCSS) in terms of concepts, relations and axioms for domain independent systems. It provides syntaxes and semantics for TCSS. Further, the transactional calculus is derived from a algebra based query language GQL-SS [11] and illustrated using examples of real life. Moreover, TCSS are proved by CAP and BASE theorems properties to show the expressiveness of the propose calculus.

II. Related Work

In previous work [11], focused on path expression in semi-structured database system. More precisely (i) described GOOSSDM [2,19,20 and 21] schema and GQL-SS [11] data are amalgamate to leaves so the path expression may carry data variables as abstractions of the content of leaves. They may also carry path variables those are evaluating to the void path or to a path having a length of n edges. The path expressions may also contain label variables to preserve labels or tags. (ii) Develop three types of algorithms. Three types of algorithms use to evaluate the path in GOOSSDM schema, one for searching return node, second for searching the path from root of GOOSSDM schema to the desired node and the third one is for the searching and listing of the tail nodes. (iii) Define the GQL-SS algebra for GOOSSDM model that operate on semi-structured schema concept and / or several constructs described in the model. The algebra consists of a set of operators and few of them can be using with the constructs like ESG, CSG separately.

As a result, point out that have to develop a transactional calculus related to this GQL-SS model. To the best of knowledge, there are no other global solutions addressing the transactional calculus for semi-structured database system. A small number of research works exist in the literatures those are in general semi-structured and used query language. However, still there is no specific transactional calculus, which is devoted enough to conceal the five challenges specified in the introduction section. The work in Supporting Multi Data Stores Applications in Cloud Environments [23] has given some idea about the semi-structured query but no proposed calculus. The amalgamation of transactions with programming control structures has provenance in systems such as Argus [28, 29].There is a composition of work that enquire into the formal specification of various zest of transactions [35, 36, 37]. However, these act of striving do not explore the semantics of transactions when integrated into a high-level programming language. Most closely related to goal is the work of Black et. al. [38], Choithia, and Duggan [39]. The former presents a theory of transactions that specify atomicity, isolation and durability properties in the form of an equivalence relation on processes. Beyond significant technical differences in the specification of the semantics, results differ most significantly from theirs insofar as [6] present a stratified semantics for a realistic kernel language intended to express different concurrency control models within the same framework. Choithia and Duggan present the pik-calculus and pike-calculus, extension of the pi calculus that supports various abstractions for distributed transactions and optimistic concurrency. Their work is relating to other efforts [40, 41] that encode transaction-style semantics into the pi-calculus and its variants. Haines et.al. [31] describes a compassable transaction facility in ML that supports persistence; undo ability, locking and threads. Their abstractions are modular and first class, although their implementation does not rely on optimistic concurrency mechanisms to handle commits. Consequently, none of the existing approaches is appropriate enough to cover the 5 challenges specified in the introduction section. In this regard, devising a new proposal, which is essential to resolve the issues, addressed in the 5 challenges.

In this case, since dealing with the combination of CAP and BASE theorem, this proposal for expressing and executing queries and real time applications shown by using the calculus. Introducing an approach for a mapping language to map attributes of the data sources to the global schema and bridge query language to write the calculus.

III. GOOSSDM: The Basic

Extending the object-oriented paradigm to semi-structured data model, the GOOSSDM introduced. It's specifying the irregular and heterogeneous structure, hierarchical and non-hierarchical relations, n – array relationships, cardinality and participation constraint of instances with all details that are required for semi-structured data model. The entire semi-structured database to be viewing as a Graph (V, E) in layered organization that is allowed by the proposed data model (GOOSSDM).At the lowest layer, each vertex represents an occurrence of an attribute or a data item.

Let consider an example of Project Management System (PMS),[11], associated with Project. Project has attributes like members, department and publications. Several members are associated with project and each member can participated in any project. Department contains member, and each individual members may have or not have publication. The PMS is semi-structured is in nature. The GOOSSDM schema for PMS has been shown in fig. 1. The sample data is showing in Table 1.

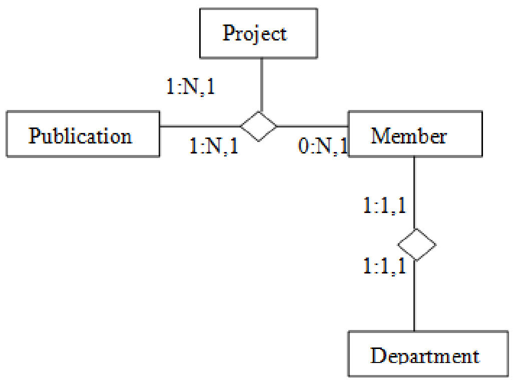


Fig.1. GOOSSDM Schema for PMS

Table 1. Sample Data Set for PMS

Project 1									
Pname	**PID**	**Topics**	**Member**			**Department**		**Publication**	
			MID	**MName**	**Maddress**	**DID**	**DeptName**	**PuID**	**Ptopics**
ABC	P1001	AAAA	M01	Bipin	XX	D01	CSE	P001	RRR
XYZ	P1003	CCCC	M03	Ashu	PP	D02	CA	P003	SSS
DEF	P1004	DDDD	M04	Rashi	YY	D03	EE	P004	TTT
XYZ	P1005	QQQQ	M06	Sashi	RR	D03	EE	P005	VVV
ABC	P1001	BBBB	M07	Priya	CC	D01	CSE	P006	MMM
Project 2									
Pname	**PID**	**Topics**	**Member**			**Department**		**Publication**	
			MID	**MName**	**Maddress**	**DID**	**DeptName**	**PuID**	**Ptopics**
PQR	P1006	YYYY	M07	Priya	CC	D02	CA	P007	NNN

IV. Calculus for Semi-Structured Database System

In previous work, defining the GQL-SS algebra for GOOSSDM model that employ on semi-structured schema impression and / or various form reportein the model. Using GOOSSDM schema the semi-structured data seen as single rooted or multi rooted graph. In every case, while initiating any query, one needs to set an immediate root for the desired CSG and then need to find the tail nodes in respect to the desired CSG.

In all the algorithms, the searching node and return node must be a type of CSG in GOOSSDM semantics. The GOOSSDM schema will use as input for the algorithms. The algorithms will invoke when the path operator (ρ) will execute. In case of proposed calculus whenever any operator will invoke, internally it will also invoke the path operator (ρ) to set the path from root node to the desired node in GOOSSDM schema by invoking algorithm 1 and algorithm 2. Moreover, the tail node list will create by invoking the algorithm 3 on next. If algorithm 1 and / or algorithm 2 return null value, then the actual operator need not to execute as there is no root available for the transactional calculus. This will facilitate the early verification of the semi-structured data schema structure in correspondence with the desired transactional calculus. The running example of Project Management System (PMS) used to illustrate the functionalities of operators. As specified earlier path operator (ρ) is also inclusive part of the algebra and invoked every time it is required to invoke any other operator specifically defined for management of semi-structured data. In the example, if Project is set as root then the path from Project to Department can be established and expressed as Project (Root)→Member→Department [11].

Algorithm 1: Searching of Node in GOOSSDM Schema

```
Step 1: Start
Step 2: Input a node C= (CSG).
Step 3: let op: = search node C
            And return node C
Step 4: let P1:=layer 0
            P2:= Immediate to layer 0
            P3:= Next to immediate layer
            P4:= Next to Next Immediate
                layer
Step 5: for i = 1 to 4
            If (op $P_i$ (C) $\neq\varnothing$) then
            Goto for next layer
            Else
            (Op $P_i$ (C))= (Root)
Step 6: stop
```

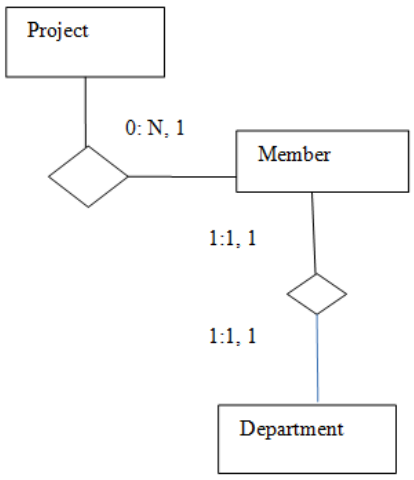


Fig.2. Searching tail node

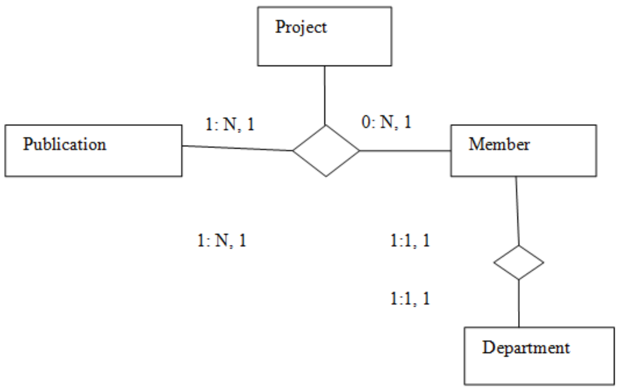


Fig.3. Searching path from root to desired node

Searching tail node from the desired node layer by layer, when the return operator of path is equal to the preceding one, then it is the last node i.e. tail node.

Algorithm 2: Searching path from Root to Desired Node

```
Step 1: Start
Step 2: Input C=CSG        // CSG for
            searching path.
Step 3: If (IS Root(C) =false) then
            N1:=op $P_i$ (CSG, P, Θ)
Step 4: If (N1==$\varnothing$) then
            Path =N1
            Else
            Goto step 3.
Step 5:  Exit
```

Root is Project, and then it searches the desired node layer by layer. Let N1is a path operator ρ with arguments layer no, CSG, and N1 value should not be Root. If N1 value is null, then the path value will be N1and if not then it will be check again from Root node.

Algorithm 3: Searching Tail Nodes from the Desired Node

```
Step 1: Start
Step 2: Input G=GOOSSDM schema
Step 3: let the path structured σ= (r,(E)), where E
is a binary relation of(CSG,P,Θ)
Step 4: For i = 1 to n                    // n= No. of
iterations
            If( IS Root(CSG))=True then
            Op<get the ith
node>(CSG,P,Θ)={$CSG_R$,$P_R$,$Θ_R$}
Step 5: for i = 1 to n
            Op< get the ithnode>( CSG,P,Θ)={$CSG_i$ ,$P_i$,$Θ_i$}
            If( (op($CSG_{i-1}$ ,$P_{i-1}$,$Θ_{i-1}$ ))== (op($CSG_i$ ,$P_i$,$Θ_i$))) then   // Finding the Tail node
            Tail =($CSG_i$ ,$P_i$,$Θ_i$)
            Else goto step 4.
Step 6 : The destination will be denoted as path
            ρ($CSG_i$ ,$P_i$,$Θ_i$)= ρ{($CSG_R$ ,$P_R$,$Θ_R$ ),
($CSG_{R-1}$ ,$P_{R-1}$,$Θ_{R-1}$),( $CSG_{R-2}$ ,$P_{R-2}$,$Θ_{R-2}$),.........,($CSG_i$, $P_i$, $Θ_i$)}
            Else goto step 3.
Step 7: Stop.
```

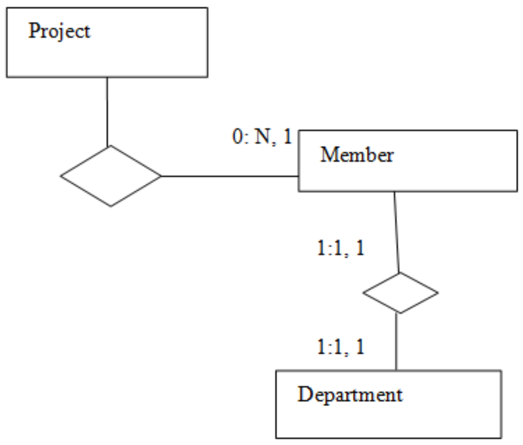


Fig.4. Searching tail node from the desired node

Searching tail node from the desired node layer by layer, when the return operator of path is equal to the preceding one, then it is the last node i.e. tail node.

A. Propose Operator

In this section, the propose operator of Transactional Calculus for Semi-structured (TCSS) of GOOSSDM model is defined. It consists of a set of operators that take one or two CSG as input and produce a new list of CSG. The fundamental operators of TCSS consist of a set of operators and few of them also can be used with constructs like CSG, ESG separately.

- *Select (σ) Operator*

The select operator will select CSG and returns CSG that satisfy a given predicate of a given list of ESGs or CSGs from the GOOSSDM schema. Thus to select those CSG from GOOSSDM schema, the tuple relational calculus (TRC) notation may be write as,

$$\{C|C \in CSG\} \tag{1}$$

Its denote that tuple C is in CSG.

$$\{C|List(C)\} \tag{2}$$

Its mean, it is the set of all tuples C such that predicate list is true for C.

[*List (CSG) =OUTPUT CSG where list= {list of ESG}*] (3)

If the set of all CSG for which the *List(C)* evaluates true. And the path expression will be like that- $\forall i: \exists C: (op\ \rho(i,C) = Root)$ [for all levels, existential CSG set the path and if it does not have any edge then it is set to Root]

$$Path(Root, C)$$
$$\forall i : Root(C) \rightarrow (op\rho(i,C))$$
$$\forall i: (op\ \rho(i,C) \leftrightarrow op\ \rho(i-1, c-1)) \rightarrow desired(C).$$

[Searching for desired CSG level by level and get ultimate CSG.] (4)

- *Retrieve (π) Operator:*

The retrieve operation allows producing the CSG from GOOSSDM schema that satisfies a given condition. The retrieve operator extracts ESG or CSG from the CSG using some constraints *CON* over one or more ESG or CSG defined in GOOSSDM schema.

$\{C|\forall C1 \in CSG(CON)\}$[*C1* belongs to some CSG with satisfied condition] (5)

It is meaning that the set of all tuples C such that for all tuples C1 is in predicate CSG is true for CON.

$$\forall C1 \exists CON\big((C1 \in CSG) \rightarrow CON(CSG)\big) \qquad (6)$$

[*C1* belongs to CSG with specified Condition and that returns the restricted CSG.]It's mean that for all tuplesC1 there exists predicate CON is true for C1is exists in CSG implies predicate

CON is true for specified CSG.

Let; *Constraints=CON*

$$CON1 = \forall C1. \exists f1$$
$$\big((C1 \in CSG) \wedge fieldname(f1) \wedge CON(C1.f1)\big) \qquad (7)$$

[The dot operator extracts ESG or CSG from the CSG using some specified constraints CON over one or more ESG or CSG defined in schema.]CON1 contains all tuples of C1 extracts the exists predicate such that C1 is exits CSG and filename (f1) and CON (C1.f1) is true.

$$CON2 = \forall C2. \exists f2((C2 \in CSG) \wedge$$
$$fieldname(f2) \wedge CON(C2.f2)) \qquad (8)$$

CON2 contains all tuples of C2 extracts the exists predicate such that C2 is exits CSG and filename (f2) and CON (C2.f2) is true.

$$\text{E.g:} \{P.Pname | Project$$
$$(P \wedge (\exists M)\big(Member(M) \wedge (M.Mid = \text{"}M07\text{"})\big)\} \qquad (9)$$

- *Union, Intersection (∪,∩)operators:*

These operators will have usual meaning. The union of any two sets *A* and *B*, denoted by A∪B, is the set of all elements which belong to *A* or *B* or both. Hence, $A \cup B = \{x: x \in A \ OR \ x \in B\}$.

$$\forall C1. \forall C2\Big(\big((C1 \in CSG) \vee (C2 \in CSG)\big) \vee CON(C1,C2)\Big) \rightarrow$$
$$(C1 \cup C2) \qquad (10)$$

[*C1* or *C2* or specify constraints of dot product of *C1* and *C2* that returns CSG or ESG which belongs to *C1* or *C2* or both.] For all C1 and C2, C1 is in exist CSG or C2 is in exist CSG or CON over both CSG implies the C1 union C2.

Intersection denoted by A∩B, is the set of elements which belong to *A* and *B* both and can be expressed as

$$A \cap B = \{ x: x \in A \ AND \ x \in B \}.$$
$$\forall C1. \forall C2\Big(\big((C1 \in CSG) \wedge (C2 \in CSG)\big) \wedge CON(C1,C2)\Big) \rightarrow$$
$$(C1 \cap C2)\,) \qquad (11)$$

[*C1* or *C2* or specified constraints of dot product of C1 and C2 that returns CSG or ESG which belongs to *C1* and *C2*.]

For all C1 and C2, C1 is in exist CSG and C2 is in exist CSG and CON over both CSG implies the C1 union C2.

- *Join (|X|) operator:*

The join operator is a special case of Cartesian product operator. It is a binary operator to relate two CSGs where one identical ESG must be common. Let, two CSGs are *CSG1* and *CSG2*. Also let, a set of ESG $E1=(E_{11}, E_{12},..., E_{1R})$ and a set of ESG $E2=(E_{21}, E_{22},..., E_{2s})$ is related with the*CSG1* and *CSG2* respectively. The join operator between *CSG1*and *CSG2* is possible iff$E1 \Lambda E2 \neq \emptyset$. Now let $E1 \Lambda E2= \{E_a, Eb, E_c\}$ then,

$$\{C1 | C1 \in CSG1 \ \Lambda \ \exists \ C2 \ (\ C2 \ (C2 \in CSG2 \ \Lambda \ C2.Ea \ = C1.Ea)\} \qquad (12)$$

[SpecifiedCSG in *C1* with Existential CSG in *C2* and both will satisfy a common ESG field.]

$$\forall C1 \exists C2 \exists E_a$$
$$\Big(\big((C1 \in CSG1) \wedge (C2 \in CSG2)\big) \rightarrow (C2.E_a = C1.E_a)\Big) \qquad (13)$$

[All CSG in *C1* and CSG in *C2* and a common ESG field is satisfied then this will return the all common ESG.]

V. Illustration of Transactional Calculus of Semi-Structured (TCSS) Database by CAP Theorem and Base Theorem

In this section, CAP theorem is as described in propose Semi-structured calculus system is as follows:

In a web concern to transmission collapse, it is difficult for any web service to execute an atomic read/write shared memory that promises a response to every request.

Proof Sketch: Having stated the CAP theorem, it is relatively straightforward to prove it correct. Consider an execution in which the nodes (servers) are partitioned into 2 disjoint set :{ N1} and (N_2 ...N_n}. Some node (client) sends a read request to server node N_2.Since N_1 is in a divergent component of the partition from N_2, every message from $N_1$to N_2 is lost. Thus it is intolerable for N_2 to differentiate the following 2 expressions:

i. There has been a preceding write of path value p1 requested of node N_1, and N_1has sent an ok response.

ii. There has been a preceding write of path value p2 requested of node N_1, and N_1has sent an ok response.

No matter how long N_2 waits, it cannot differentiate these 2 cases, and as a consequence it cannot ascertain whether to return response p1 or p2. Server node N_2 eventually must return a response, even if the system is segregated; if the message delay from N_1 to N_2 is sufficiently large that N_2 believes the system to be differentiated, then it may return an erroneous response, despite the scarcity of partitions.

The paramount explanation for extending the CAP theorem is to make the point that in the majority of instances, a distributed system can only guarantee two of the features, not all three. To ignore such a decision could have catastrophic results that include the possibility of all three elements falling apart simultaneously.

Consistency: A read sees all previously completed writes i.e. all nodes see the same data at the same time .g: As the above figure I show that, if Project is set as Root then the Path from Project to department can be established and expressed as ; *Project (Root)→Member→Department.*
Let; the path denoted as *ρ.*
Then, it can be expressed as-
ρ(R,C)= the path from *Root to CSG.*
Root denoted as *R, C(CSG) and E is a* trinary relation of*(CSG,P,Θ)*

$$\therefore \forall i{:}\, \rho(i)[\rho\ is\ the\ layer]$$
$$And \forall i{:}\, \exists C{:}\, (op\rho(i,C) = Root) Then, \forall i:\ Root(C) \rightarrow [op\rho(i,C)] \tag{14}$$

$$\forall i{:}\, [op\rho(i,C) \leftrightarrow op\rho(i-1,c-1)] \rightarrow Tail(C). \tag{15}$$

$$\therefore \rho 1(R,C) \vee \rho 2(R,C) \ldots\ldots\ldots \vee \rho n(R,C) \rightarrow \rho(R,C). \tag{16}$$

For all i ,*ρ* satisfies the layer, for all i and existential C if operator *ρ* with layer and CSG is satisfied Root then Root implies the operator *ρ* with layer and CSG. If the operator *ρ* with layer and CSG satisfies the preceding layer and CSG then it implies the tail node.

Therefore, all nodes see the same data at the same time. In addition, it also satisfy the Base Theorem Basic Availability that means it response to any request.

Availability: Guarantee that every request receives a response about whether it succeeded or failed i.e. read and write always succeed. This means that in GOOSSDM schema there should be a searching path and its return some value. The path value should not be null. Here defining a path means it guarantees that every request receives a response about whether it succeeded or failed i.e. read and write always succeed. When it succeeded then it is succeeded path otherwise, it is failed path.

Succeeded path =N1
Failed Path or$\neg(\mathrm{Succeededpath}) = \mathrm{N1}$

$$\therefore \forall i{:}\, \rho(i)\ [\rho\ is\ the\ layer]$$
$$And \forall i{:}\, \exists C{:}\, (op\rho(i,C) = Root) \tag{17}$$

$$C(CSG)[Let;\ N1{:}= op\rho(i,C)] \tag{18}$$

$$\exists C{:}\ \neg Root(C)\ \rightarrow N1. \tag{19}$$

$$\big(\mathrm{Succeededpath(N1)}\big) \rightarrow [\exists N1{:}\, N1(\phi) \rightarrow Path(N1)] \tag{20}$$

$$\exists N1{:}\, \big((N1(\phi) \rightarrow Path(N1)\big) \Lambda \Big(\neg N1(\phi) \rightarrow Path(N1) \Lambda \big(\neg N1 = (\phi)\big)\Big) \tag{21}$$

For existential C, let succeeded path is not root. Succeeded path implies for existential N1 if N1 value is null then this will be the path value, should not be null, then again for existential N1 returns failed path or succeeded path or succeeded path with not null value.

Therefore, all searching path must return some value. Again, it is also satisfying the Base Theorem Soft State that according to the users' requirement the desired path will change and it must return some value.

Partition Tolerance: Guaranteed properties are maintained even when network failures prevent some machines from communicating with others. The system continues to operate despite arbitrary partitioning due to network failures.

$$Path(Root\ ,E)\ \ [E\ is\ a\ trinary\ relation\ of\ (CSG,P,\Theta)]$$
$$\forall i:\ Root(C) \rightarrow [opp(i,C)] \tag{22}$$

$$\forall i{:}\, [opp(i,C) \leftrightarrow opp(i-1,c-1)] \rightarrow Tail(C) \tag{23}$$

$$Path1(Root,E) V Path2(Root,E) V \ldots Pathn(Root,E) \rightarrow Path(Root,E). \tag{24}$$

For all i, the Root implies operator *ρ* with layer and CSG. If the operator *ρ* with layer and CSG satisfies the preceding layer and CSG then it implies the tail node. The all-possible paths of OR operation implies the desired node.

Therefore, the every Node will cultivate q to everywhere it should sooner or later, but the path will continue to receive input and is not checking the consistency of every transaction before it moves onto the next node.

Read-Write Operation Algorithm

Assuming node R is the Root node. The algorithm behaves as follows and A is desired node.

Algorithm 1: Read at node A

Step 1: A sends a request to R for the recent value.
Step 2: If A receives a response from R that means find a path value, then save the value and send it to \\\the client.

By applying algorithm R is the root node and scanning from R to the desired node, A returns the path value with arguments in operator layer no and CSG and it is the finding of path value.

Algorithm 2: Write at node A

Step 1: A sends a message to R with the new path value.

Step 2: If A receives an ACK from R, then A sends an ACK to the client and stop.
Step 3: If A has not yet received an ACK from R, then A sends a message to R with the new value.

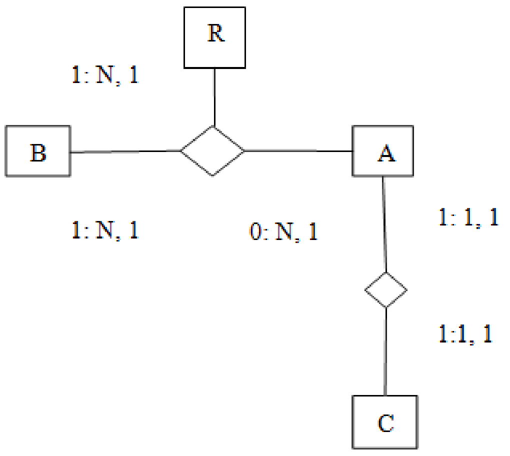


Fig.5. Example of read at node A

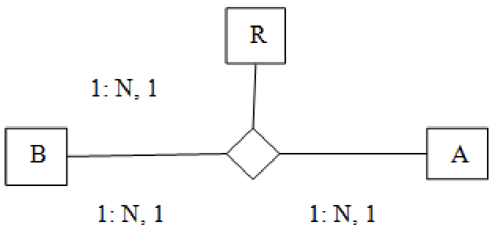


Fig.6. Example of write at node A

A sends request to R for the new path value and R scans it from right to left, i.e. R→B→A; A have to wait to get the ACK and B will get the ACK prior to A and then A sends a message to R with the new value.

Algorithm 3: New value is receiving at node R

Step 1: R increments its sequence no by 1.
Step 2: R sends out the new value and sequence no to every node.

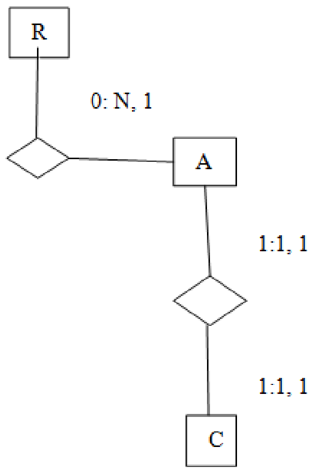


Fig.7. Example of New value is received at node R.

According to previous algorithm Root will increment its layer value by 1 and every node will getting there layer no i.e. sequence no.

VI. Validation of Transactional Calculus of Semi-structured (TCSS) database by CAP and Base Theorem

Data validation intended to provide certain well-defined guarantees for fitness, accuracy, and consistency for any of various kinds of user input into an application or automated system. Data validation rules can be defined and designed using any of various methodologies, and be deployed in any of various contexts.

Data validation, as explained above, is making sure that all data (whether user input variables, read from file or read from a database) are valid for their intended data types and stay valid throughout the application that is driving this data. What this means is data validation, in order to be as successful as it can be, must implemented at all parts that get the data, processes it and saves or prints the results.

Validation

In evaluating the basics of data validation, generalizations can made regarding the different types of validation, according to the scope, complexity, and purpose of the various validation operations to be carried out. For example:

Data type validation: Data type validation customarily carried out on one or more simple data fields. The simplest kind of data type validation verifies that the individual characters provided through user input are consistent with the expected characters of one or more known primitive data types; as defined in a programming language or data storage and retrieval mechanism. As the above figure I show that, if Project are set as Root then the Path from Project to Department can be established and expressed as ; ***Project (Root)→Member→Department.***
Let ; the path denoted as ρ.
Then , it can be expressed as-

$\rho(R,C)$=i.e. the path from *Root to CSG.*

$$\therefore \forall i: \rho(i)\ [\rho\ is\ the\ layer]$$

$$And\quad \forall i: \exists C: (op\ \rho(i,C) = Root) \tag{25}$$

$$Then, \forall i:\ Root(C) \rightarrow [op\ \rho(i,C)] \tag{26}$$

$$\forall i: [op\ \rho(i,C) \leftrightarrow op\ \rho(i-1,c-1)] \rightarrow Tail(C). \tag{27}$$

$$\therefore (valid(path) \rightarrow (\rho 1(R,C) V \rho 2(R,C) \ldots\ldots.. V \rho n(R,C) \rightarrow \rho(R,c)). \tag{28}$$

This is the simple example of data validation that verifies that the individual characters provided through user input are consistent with the expected characters of one or more known primitive data types; as defined in a programming language or data storage and retrieval mechanism and in previous section it is already proved that it satisfy the CAP and BASE Theorem.

Constraint validation: Constraint validation may examine user input for consistency with a minimum/maximum range, or consistency with a test for evaluating a sequence of characters,

Consistency: A read sees all previously completed writes i.e. all nodes see the same data at the same time.
E .g: As the above figure I show that, if Project are set as Root then the Path from Project to department can be established and expressed as ; *Project (Root)→Member→Department.*
Let ; the path denoted as *ρ.*
Then , it can be expressed as-
*ρ(R,C)=*i.e. the path from *Root to CSG.*
Root denoted as *R, C(CSG) and E* trinary relation of*(CSG,P,Θ)*

$$\therefore \forall i{:}\rho(i)\ [\rho \text{ is the layer}]$$
$$And\ \forall i{:}\, \exists C{:}\, (op\ \rho(i,C) = Root) \qquad (29)$$

$$Then, \forall i:\ Root(C) \rightarrow [op\ \rho(i,C)] \qquad (30)$$

$$\forall i{:}\, [opp(i,C) \leftrightarrow opp(i-1,c-1)] \rightarrow Tail(C). \qquad (31)$$

$$\therefore \rho 1(R,C) V \rho 2(R,C) \ldots\ldots. V \rho n(R,C) \rightarrow \rho(R,c). \qquad (32)$$

Therefore, all nodes see the same data at the same time. This is the simple example of constraint validation and in constraint validation examine for consistency. In previous section it is already proved that consistency satisfy the CAP and BASE Theorem.

Structured validation: Structured validation allows for the combination of any of various basic data-type validation steps, along with more complex processing. Such complex processing may include the testing of conditional constraints for an entire complex data object or set of process operations within a system.

Path(Root ,E) [C(CSG) and E trinary relation of*(CSG,P,Θ)]*

$$\forall i:\ Root(C) \rightarrow [op\ \rho(i,C)] \qquad (33)$$

$$\forall i{:}\, [op\ p(i,C) \leftrightarrow op\ \rho(i-1,c-1)] \rightarrow Tail(C) \qquad (34)$$

$$Path1(Root,E) V Path2(Root,E) V \ldots Pathn(Root,E) \rightarrow Path(Root,E). \qquad (35)$$

Therefore, the every Node will propagate to everywhere it should sooner or later, but the path will continue to receive input.

This is the example of Structured validation it include complex processing such complex processing may include the testing of conditional constraints for an entire complex data object or set of process operations within a system.

VII. TCSS Operators with Example

In previous work defining the GQL-SS algebra for GOOSSDM model that operate on semi-structured schema concept and / or several constructs described in the model. The algebra consists of a set of operators and few of them also can be used with the constructs like ESG, CSG separately. The running example of *Project Management System (PMS)* used to illustrate the functionalities of operators. As specified earlier *path operator (ρ)* is also inclusive part of the algebra and invoked every time it is required to invoke any other operator specifically defined for management of semi-structured data.

Let consider an example of Project Management System (PMS) where a project has several members and members are associated with some departments. Individual members either may or may not have publications. Moreover, each member may participate in any number of projects. The database for PMS is purely semi-structured in nature. The sample data has been showing in table I.

A. Operators in GOOSSDM

Let us note that in GOOSSDM the data are seen as single rooted graphs or multi rooted graph. In every cases have to set an immediate root for the desired CSG and then also find the tail node in respect to the desired CSG.

- ***Select (σ) Operator:*** The select operator will select CSG and returns CSG that satisfy a given list of ESGs or CSGs from the GOOSSDM schema. The tuple relational calculus (TRC) notation is,

$$\{C|C \in CSG\} \qquad (36)$$

$$\{C|List(C)\} \qquad (37)$$

[List(CSG)=OUTPUT CSG where list={list of ESG}]

If the set of all CSG for which the List(C) evaluates true. And the path expression will be like that-

$$\forall i{:}\, \exists C{:}\, (op\ \rho(i,C) = Root) \qquad (38)$$

[for all levels, existential CSG set the path and if it does not have any edge then it is set to Root]

$$Path(Root, C) \qquad (39)$$

$$\forall i:\ Root(C) \rightarrow (op\rho(i,C)\ ) \qquad (40)$$

$$\forall i{:}\, \big(op\ \rho(i,C) \leftrightarrow op\ \rho(i-1,c-1)\big) \rightarrow desired(C). \qquad (41)$$

[Searching for desired CSG level by level and get ultimate CSG.]

- ***Retrieve (π) Operator:*** The retrieve operator extracts ESG or CSG from the CSG using some constraints *CON* over one or more ESG or CSG defined in GOOSSDM schema.

$$\{C|\forall C1 \in CSG(CON)\} \quad (42)$$

[C1 belongs to some CSG with satisfied condition]

$$\forall C1\exists CON\big((C1 \in CSG) \rightarrow CON(CSG)\big) \quad (43)$$

[C1 belongs to CSG with specified condition and that returns the restricted CSG.]Let; Constraints=CON

$$CON1 = \forall C1.\exists f1\big((C1 \in CSG) \wedge fieldname(f1) \wedge CON(C1.f1)\big) \quad (44)$$

[The dot operator extracts ESG or CSG from the CSG using some specified constraints CON over one or more ESG or CSG defined in schema.]

$$CON2 = \forall C2.\exists f2((C2 \in CSG) \wedge fieldname(f2) \wedge CON(C2.f2)) \quad (45)$$

$$E.g: \{P.Pname|Project(P \wedge (\exists M)\big(Member(M) \wedge (M.Mid = \text{"M07"})\big)\} \quad (46)$$

- ***Union, Intersection and Difference (∪,∩,and -)operators*:** These operators will have usual meaning. The union of any two sets *A* and *B*, denoted by A∪B, is the set of all elements which belong to *A* or *B* or both. Hence, $A \cup B =\{ x: x \in A \ OR\ x \in B\}$.

$$\forall C1.\forall C2\Big(\big((C1 \in CSG)\wedge(C2 \in CSG)\big) \wedge CON(C1, C2)\Big) \rightarrow (C1 \cup C2) \quad (47)$$

[C1 or C2 or specified constraints of dot product of C1 and C2 that returns CSG or ESG which belongs to C1 or C2 or both.]

Intersection denoted by A∩B, is the set of elements, which belong to A, and *B* both, expressed as

$$A\cap B= \{ x: x\in A\ AND\ x\in B \}.$$

$$\forall C1.\forall C2\Big(\big((C1 \in CSG)\wedge(C2 \in CSG)\big) \wedge CON(C1, C2)\Big) \rightarrow (C1 \cap C2)\ ) \quad (48)$$

[C1 or C2 or specified constraints of dot product of C1 and C2 that returns CSG or ESG which belongs to C1 and C2.]

- ***Join (|X|) operator:*** The join operator is a special case of Cartesian Product operator. It is a binary operator to relate two CSGs where one identical ESG must be common. Let, two CSGs are *CSG1* and *CSG2*. Also let, a set of ESG $E1= (E_{11}, E_{12}... E_{1R})$ and a set of ESG $E2= (E21, E_{22}... E_{2s})$ is related with the*CSG1* and *CSG2* respectively. The join operator between *CSG1*and *CSG2* is possible iff$E1\Lambda E2\neq\emptyset$. Now let$E1\Lambda E2=\{E_a,E_b, E_c\}$ then,

$$\{C1|C1\in CSG1 \wedge \exists\ C2\ (\ C2\ (C2\in CSG2 \wedge C2.Ea\ = C1.Ea)\} \quad (49)$$

[Specified CSG in C1 with Existential CSG in C2 and both will satisfy a common ESG field.]

$$\forall C1\exists C2\exists E_a\Big(\big((C1 \in CSG1) \wedge (C2 \in CSG2)\big) \rightarrow (C2.E_a = C1.E_a)\Big) \quad (50)$$

[All CSG in C1 and CSG in C2 and a common ESG field is satisfied then this will return the all common ESG.]

B. Capabilities of the proposed calculus TCSS

In this section, the expressiveness capabilities of the proposed calculus of TCSS demonstrated by applying the tuple relational calculus to suitable example queries.

a. Find the project name and project id from the CSG project1.

In this query, the *Select* operator has been using to select list like *Pname* and *PID* from *Project1*.The calculus can be expressed as follows,

{P.Pname, P.PID|Project1 (P)}.

Result:

```
<Project1>
<PName> ABC</PName>
<PID>P1001</PID>
<PName> XYZ</PName>
<PID>P1003</PID>
<PName> DEF</PName>
<PID>P1004</PID>
<PName> XYZ</PName>
<PID>P1005</PID>
<PName>ABC</PName>
<PID>P1001</PID>
</Project1>
```

b. Find the details of publication whose Member Id MID="M03" and Publication Id PuID="P003".

In this query, the *Retrieve* operator has been used with the constraints of *select* operation on select list as*MID ="M03"* from *Member* CSG and also select the list as*PID="P003"* from *Publication* CSG. The calculus can be expressed as follows,

{P.Publication|Project1(p)Λ(∃)(Member(M)ΛM.MID=' M03')Λ(∃)(Publication(B)ΛB.PuID='P003')}

Result:
```
<Project1>
<Publication>
<PuID> P003 </PuID>
<Ptopics>SSS</Ptopics>
</Publication>
</Project1>
```

c. Find the details of member where MName=”Bipin” from project1 and also find the details of Member where MName=”Priya” from Project2.

In this query, the *Retrieve* operator has been used with the constraints of *select* operation on the list *Mname =”Bipin”*and*Mname =”Priya”* from *Member* CSG. The calculus can be expressed as follows,

{P.Member|Project1(P)Λ(∃)(Member(M)ΛM.MName=’Bipin’)}V{P.Member|Project2(P)Λ(∃)(Member(M)ΛM.MName=’Priya’}

Result:
```
<Project1>
<Member>
<MID> M01</MID>
<MName>Bipin</MName>
<MAddress> XX </MAddress>
</Member>
</Project1>
<Project2>
<Member>
<MID> M07</MID>
<MName>Priya</MName>
<MAddress> CC </MAddress>
</Member>
</Project2>
```

d. Find the name of all members who have the same department id “DID=D03” and department name “EE”.

In this query, the *Retrieve* operator has been used with the constraints of *select* operation as the list *DID=”D03”* from *Department* CSG. Also another *Retrieve* operator has been used with constraints on *select* operation as the list *DName=”Electrical”* from *Department* CSG. Finally the intersection operator has been used. The calculus can be expressed as follows

{P.Member|Project1(P)Λ(∀□)(Member(M))Λ(∃)(Department(D)ΛD.DID=’D03’ΛD.Dname=’EE’} Result:

```
<Project1>
<Member>
<MName>Rashi</MName>
</Member>
<Member>
<MName>Sashi</MName>
</Member>
</Project1>
```

e. Find the name of the all members who have the department id same.

In this query, required to set the custom root and then required to apply the join operator. For the purpose, the*Member*CSG needs to set the root. The calculus can be expressed by semantics and corresponding result are as follows

{P.Member|Project1(P)Λ(∀) (M)((Member(M))Λ(Department(D))→ D.DID=D.DID)}

Result:
```
<Member>
<MName>Bipin</MName>
<MName>Rashi</MName>
<MName>Sashi</MName>
<MName>Priya</MName>
</Member>
```

f. Find the project name and project id from the CSG project1 and CSG project2.

In this query, the *Select* operator has been used to select list like *Pname* and *PID* from *Project1 and also Project2*.The calculus can be expressed as follows:

{P.Pname,P.PID|Project1(P)}.V{P.Pname,P.PID|Project2(P)}.

Result:
```
<Project1>
<PName> ABC</PName>
<PID>P1001</PID>
<PName> XYZ</PName>
<PID>P1003</PID>
<PName> DEF</PName>
<PID>P1004</PID>
<PName> XYZ</PName>
<PID>P1005</PID>
<PName>ABC</PName>
<PID>P1001</PID>
</Project1>
<Project2>
<PName> ABC</PName>
<PID>P1001</PID>
<Project2>
```

g. Find the details of publications where MName=”Bipin” from project1 and also find the details of publication where MName=”Priya” from Project2.

In this query, the *Retrieve* operator has been used with the constraints of *select* operation on the list *Mname =”Bipin”*and*Mname =”Priya”* from *Member* CSG. The calculus can be expressed as follows

{P.Publication|Project1(P)Λ(∃)(Member(M)ΛM.MName=’Bipin’)}V{P.Publication|Project2(P)Λ(∃)(Member(M)ΛM.MName=’Priya’}

Result:
```
<publication>
<puid> P001</puid>
<ptopics> RRR </ptopics>
<puid> P007</puid>
```

```
<ptopics> NNN</ptopics>
</publication>
```

VIII. An Implementation of proposed TCSS

A. *Transaction Execution:*

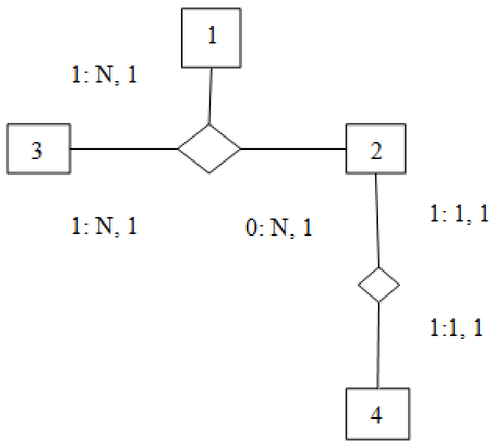


Fig.8. Example of transaction execution

The above figure 8 shows the root node is 1 and then scanning from right, the next node is 2 and the next after next node is 4 after that it scans for the left node 3.Focusing on a simplified variant of TCSS, that is dynamically typed. To introduce the syntaxes and semantics of TCSS, let us starting with a simple example of transactional query by using x-query. In this section, the expressiveness capabilities of the proposed transactional calculus of TCSS demonstrated by applying the calculus to suitable example queries.

```
<project>
<project1>
<pname>ABC</pname>
<pid>P1001</pid>
<topics>AAAA</topics>
<member>
<mid>M01</mid>
<mname>BIPIN</mname>
<maddress>xx</maddress>
<department>
<did>D01</did>
<dname>CSE</dname>
<publication>
<puid>P001</puid>
<ptopics>RRR</ptopics>
</publication>
</department>
</member>
<pname>XYZ</pname>
<pid>P1003</pid>
<topics>CCCC</topics>
<member>
<mid>M03</mid>
<mname>ASHU</mname>
<maddress>PP</maddress>
<department>
<did>D02</did>
<dname>CA</dname>
<publication>
<puid>P003</puid>
<ptopics>SSS</ptopics>
</publication>
</department>
</member>
<pname>DEF</pname>
<pid>P1004</pid>
<topics>DDDD</topics>
<member>
<mid>M04</mid>
<mname>RASHI</mname>
<maddress>YY</maddress>
<department>
<did>D03</did>
<dname>EE</dname>
<publication>
<puid>P004</puid>
<ptopics>TTT</ptopics>
</publication>
</department>
</member>
<pname>XYZ</pname>
<pid>P1005</pid>
<topics>QQQQ</topics>
<member>
<mid>M06</mid>
<mname>SASHI</mname>
<maddress>RR</maddress>
<department>
<did>D03</did>
<dname>EE</dname>
<publication>
<puid>P005</puid>
<ptopics>VVV</ptopics>
</publication>
</department>
</member>
<pname>ABC</pname>
<pid>P1001</pid>
<topics>BBBB</topics>
<member>
<mid>M07</mid>
<mname>PRIYA</mname>
<mid>M07</mid>
<mname>PRIYA</mname>
<maddress>CC</maddress>
<department>
<did>D01</did>
<dname>CSE</dname>
<publication>
<puid>P006</puid>
<ptopics>MMM</ptopics>
</publication>
</department>
</member>
</project1>
<project2>
<pname>PQR</pname>
<pid>P1006</pid>
<topics>YYYY</topics>
<member>
<mid>M07</mid>
<mname>PRIYA</mname>
<maddress>cc</maddress>
<department>
<did>D02</did>
<dname>CA</dname>
 -<publication>
<puid>P007</puid>
<ptopics>NNN</ptopics>
</publication>
</department>
</member>
</project2>
</project>
```

1. ***Find the project name and project id from the CSG project1.***

```
for $p1 in doc("demo1.xml")//project1
for $p2 in doc("demo1.xml")//project1
where $p1//topics != $p2//topics
return<table ID="project">
<pname>{data($p1//pname)}</pname>
<pid>{data($p1//pid)}</pid>
</project>
</table>
```

```
<table ID=" project">
<pname> ABC  XYZ  DEF  XYZ ABC
</pname>
<pid> P1001  P1003  P1004  P1005  P1001
</pid>
</project>
</table>
```

2. ***Find the details of publication whose Member Id MID="M03" and Publication Id PID="P003".***

```
for $p in doc("demo1.xml")//member
where $p//mid = "M03"
and $p//puid = "P003"
return $p//publication
<publication>
<puid> P003 </puid>
<ptopics> SSS</ptopics>
</publication>
```

3. ***Find the details of member where MName="Bipin" from project1 and also find the details of Member where MName="Priya" from Project2.***

```
for $p1 in doc("demo.xml")/project/project1/member
for $p2 in doc("demo.xml")/project/project2/member
where $p1//mname = "BIPIN"
and $p2//mname = "PRIYA"
return<table ID= "project">
<member>
{$p1//(mid,mname,maddress)}
{$p2//(mid,mname,maddress)}
</member>
</table>
< table ID= "project">
<member>
  <mid> M01</mid>
  <mname> BIPIN </mname>
  <maddress> XX </maddress>
  <mid> M07</mid>
  <mname> PRIYA</mname>
  <maddress> CC</maddress>
</member>
</project>
</table>
```

4. ***Find the name of all members who have the same department id "DID=D03" and department name "EE".***

```
for $p in doc("demo1.xml")//member
where $p//dname = "EE"
and $p//did = "D03"
return<project1>
<member>
{$p//(mid,mname,maddress)}
</member>
</project1>
 <project1>
   <member>
     <mname> RASHI </mname>
   </member>
 </project1>
 <project1>
   <member>
     <mname> SASHI </mname>
   </member>
 </project1>
```

5. ***Find the name of the all members who have the department id same***

```
for $p1 in doc("demo1.xml")/project/project1/member
for $p2 in doc("demo1.xml")/project/project1/member
where $p1//did = $p2//did
and $p1//puid != $p2//puid
return<member>
<mname>{data($p1//mname)}</mname>
</member>
   <member>
  <mname> BI PIN</mname>
</member>
<member>
  <mname> RASHI </mname>
</member>
<member>
   <mname> SASHI </mname>
</member>
 <member>
   <mname> PRIYA </mname>
 </member>
```

6. ***Find the project name and project id from the CSG Project1 and Project2***

```
for $p1 in doc("demo1.xml")//project1
for $p2 in doc("demo1.xml")//project
where $p1//topics != $p2//topics
return<table ID="project">
<pname>{data($p1//pname)}</pname>
<pid>{data($p1//pid)}</pid>
<pname>{data($p2//pname)}</pname>
<pid>{data($p2//pid)}</pid>
</table>
< table ID="project">
  <pname>ABC XYZ DEF XYZ ABC</pname>
   <pid> P1001 P1003 P1004 P1005 P1001</pid>
  <pname> PQR</pname>
  <pid>  P1006</pid>
</table>
```

7. ***Find the details of publications where MName="Bipin" from project1 and also find the details of publication where MName="Priya" from Project2.***

```
for $p1 in doc("demo1.xml")/project/project1/member
for $p2 in doc("demo1.xml")/project/project2/member
where $p1//mname = "BIPIN"
and $p2//mname = "PRIYA"
return<table ID= "project">
<publication>
{$p1//(puid,ptopics)}
{$p2//(puid,ptopics)}
</publication>
</table>
<table ID="project">
  <publication>
    <puid> P001</puid>
    <ptopics> RRR </ptopics>
     <puid> P007</puid>
     <ptopics> NNN</ptopics>
</publication>
</table>
```

B. Implementation of TCSS X-Query

To examine the scalability of proposed TCSS X-Query implementation, trying to perform an experimental evaluation using "Project" xml data. Here also trying to perform a comparison of TCSS X-Query with open source xml processors: BASE-X.

Queries

Here considering 5 basic types of queries: Selection, Retrieve, Union, Intersection and Join.

Selection: Query 1 finds the project name and project id from the CSG project1

```
for $p1 in doc("demo1.xml")//project1
for $p2 in doc("demo1.xml")//project1
where $p1//topics != $p2//topics
return<table ID="project">
<pname>{data($p1//pname)}</pname>
<pid>{data($p1//pid)}</pid>
</project>
</table>
```

Query 1

Retrieve: Query 2 finds the details of publication whose Member Id MID="M03" and Publication Id PID="P003".

```
for $p in doc("demo1.xml")//member
where $p//mid = "M03"
and $p//puid = "P003"
return $p//publication
```

Query 2

Union: Query 3 finds the details of member where MName="Bipin" from project1 and also find the details of Member where MName="Priya" from Project2.

```
for $p1 in
doc("demo.xml")/project/project1/member
for $p2 in
doc("demo.xml")/project/project2/member
where $p1//mname = "BIPIN"
and $p2//mname = "PRIYA"
return<table ID= "project">
<member>
{$p1//(mid,mname,maddress)}
{$p2//(mid,mname,maddress)}
</member>
</table>
                                     Query 3
```

Intersection: Query 4 finds the name of all members who have the same department id "DID=D03" and department name="EE".

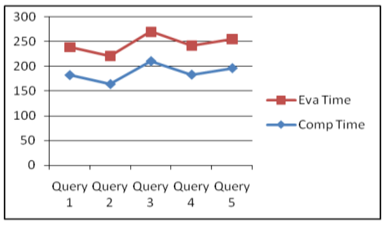
```
for $p in doc("demo1.xml")//member
where $p//dname = "EE"
and $p//did = "D03"
return<project1>
<member>
{$p//(mid,mname,maddress)}
</member>
</project1>
                                     Query 4
```

Join: Query 5finds the name of the all members who have the department id same.

```
for $p1 in
doc("demo1.xml")/project/project1/member
for $p2 in
doc("demo1.xml")/project/project1/member
where $p1//did = $p2//did
and $p1//puid != $p2//puid
return<member>
<mname>{data($p1//mname)}</mname>
        </member>
                                     Query 5
```

C. Experimental Results

This paper performance study explores TCSS X-Query ability. Here in Fig 9 it shows that in case of TCSS X-Query each query execution time is near about same to each other and its maintain a parity, whereas BASE-X x-query processor takes more time for selection procedure and takes less time for join queries. Whereas TCSS x-query time remains comparable, i.e. the additional data is processing in the same amount of time. Here TCSS X-Query demonstrated using a real 10 KB XML dataset (trying to perform an experimental evaluation using "Project" xml data.') for various XML selection, retrieve, union, and intersection and join queries. In future, planning to analysing of big xml data and optimization of the query compiler.



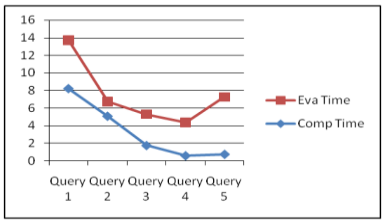


Fig.9. above TCSS X-Query and below BASE-X X-Query

VIII. CONCLUSION

The proposed framework blends semantic of transactional calculus specification and abstraction mechanism with syntaxes in specific modelling. Thus, the paper fulfils the deficiency of systematic methodology in transactional calculus of GOOSSDM model. In addition to this paper proposes a formal transactional calculus called Transactional Calculus for Semi-structured database (TCSS) Further, the transactional calculus is derived from a algebra based query language [11] and illustrated using examples of real life. The benefits of the proposed work are manifold. It provides supports towards (1) representation of precise knowledge of domain independent conceptualisation from structural and functional design concerns with enriched semantics and syntaxes for transactional calculus of semi-structured. (2) realisation of proposed TCSS working with CAP and BASE theorem. (3) a systematic methodology that pave the way of transforming domain analysis. (4) providing guidelines for the purpose of mapping of Transactional Calculus for Semi-structured database. (5) the proposed Transactional system for semi-structured is based on path expression. (6) the path operator is used to set the root node in GOOSSDM schema and also useful to find the path from the root node to desired node for any transaction. (7) facilitate the early verification of the semi-structured data schema structure in correspondence with the desired transactional calculus. The perspective is an extension to this calculus allowing to support larger class of complex queries like aggregates, group by operations.

REFERENCES

[1] Conrad R., Scheffner D., Freytag J. C., "XML conceptual modeling using UML", 19thIntl. Conf. on Conceptual Modeling, PP: 558-574, 2000.

[2] Anirban Sarkar, “Design of Semi-structured Database System: Conceptual Model to Logical Representation”, Book Titled: Designing, Engineering, and Analyzing Reliable and Efficient Software, Editors: H. Singh and K. Kaur, IGI Global Publications, USA, PP 74 – 95, 2013.

[3] McHugh J., Abiteboul S., Goldman R., Quass D., Widom J., "Lore: a database management system for semistructured data", Vol. 26 (3), PP: 54 - 66, 1997.

[4] Badia, A., "Conceptual modeling for semistructured data", 3rdInternational Conference on Web Information Systems Engineering, PP: 170 – 177, 2002.

[5] Mani M., “EReX: A Conceptual Model for XML”, 2ndInternational XML Database Symposium, PP 128-142, 2004.

[6] Suresh Jagannathan, Jan Vitek,Adam Welc, Antony Hosking, A Transactional Object Calculus, Dept of Comp.sc,Purdue University, West Lafayette, IN 47906, United States.

[7] Liu H., Lu Y., Yang Q., "XML conceptual modeling with XUML", 28thInternational Conference on Software Engineering, PP: 973–976, 2006.

[8] Combi C., Oliboni B., "Conceptual modeling of XMLdata", ACM Symposium on Applied Computing, PP: 467– 473, 2006.

[9] Wu X., Ling T. W., Lee M. L., Dobbie G.," Designing semistructured databases using ORA-SSmodel", 2ndInternational Conference on Web Information Systems Engineering, Vol. 1, PP: 171 –180, 2001.

[10] Seth Gilbert and Nancy Lynch. Brewer’s conjecture and the feasibility of consistentavailable, partition tolerant web services.SigActNews, June2002.

[11] Rita Ganguly, RajibKumarchatterjee,Anirban Sarkar. “Graph Semantic based Approach for Quering Semi-structured Database System.”22nd International Conference on SEDE-2013, pp: 79-84.

[12] Seth Gilbert National University of Singapore and Nancy Lynch. Brewer’sMassachusetts Institute of Technology,”Perspectives on the CAP Theorem.

[13] Soichiro Hidaka Zhenjiang Hu Kazuhiro Inaba Hiroyuki Kato , “Bidirectionalizing Structural Recursion on Graphs”,Techical Report, National Institute of Informatics, The University of Tokyo/JSPS Research Fellow, The University of Electro-Communications, August 31, 2009

[14] Data Validation, Data Integrity, Designing Distributed Applications with Visual Studio NET, Arkady Maydanchik (2007), "Data Quality Assessment", Technics Publications, LLC

[15] Object Oriented Transaction Processing in the KeyKOS® Microkernel. William S. Frantz ,Periwinkle Computer Consulting, 16345 Englewood Ave. Los Gatos, CA USA 95032 rantz@netcom.com Charles R. Landau ,Tandem Computers Inc. 19333

[16] Vallco Pkwy, Loc 3-22 ,Cupertino, CA USA 95014 landau_charles@tandem.com. Introduction to Object-Oriented Databases. Prof. Kazimierz Subieta ,subieta@pjwstk.edu.pl,http://www.ipipan.waw.pl /~subieta Ni W., Ling T. W., “GLASS: A Graphical Query Language for Semi-structured Data”, 8th International Conference on Database Systems for Advanced Applications, PP 363 –370, 2003.

[17] R. K. Lomotey and R. Deters, “Datamining from document-append NoSQL,” Int. J. Services Comput., vol. 2, no. 2, pp. 17–29, 2014.

[18] Braga, D., Campi, A. and Ceri, S., “XQBE (XQuery By Example): A visual interface to the standard XML query language”, ACM Transactions on Database Systems(TODS), Vol.30 (5), pp. 398 – 443, 2003.

[19] AnirbanSarkar, "Conceptual Level Design of Semi-structured Database System: Graph-semantic Based Approach", International Journal of Advanced Computer Science and Applications, The SAI Pubs. , New York, USA, Vol. 2, Issue 10, PP 112 – 121,November, 2011. [ISSN: 2156-5570(Online) &ISSN : 2158-107X(Print)].

[20] T. W. Ling. A normal form for sets of not-necessarily normalized relations. In Proceedings of the 22nd Hawaii International Conference on System Sciences, pp. 578-586. United States: IEEE Computer Society Press, 1989.

[21] T. W. Ling and L. L. Yan. NF-NR: A Practical Normal Form for Nested Relations. Journal of Systems Integration. Vol4, 1994, pp309-340.

[22] Rita Ganguly,Anirban Sarkar “ Evaluations of Conceptual Models for Semi-structured Database system “. International Journal of ComputerApplications.Vol 50, Issue 18, PP 5-12,july,2012.[ISBN:973-93-80869-67-3].

[23] Rami Sellami, Sami Bhiri , and Bruno Defude, “Supporting Multi Data Stores Applications in cloud Environments.” IEEE Transactions on services computing, vol-9, No-1,pp-59-71, January/February2016.

[24] O. Cur_e, R. Hecht, C. Le Duc, and M. Lamolle, “Data integration over NoSQL stores using access path based mappings,” inProc. 22nd Int. Conf. Database Expert Syst. Appl., Part I, 2011, pp. 481–495.

[25] ACID vs. BASE: The Shifting pH of Database Transaction Processing, By Charles Roe, www.dataversity net

[26] Martin Abadi Microsoft Research.university of california santa cruz, Tim Harris, Microsoft Research, Katherine F Moore Microsoft Research, University of Washington, “A Model of Dynamic Seperation for Transactional Memory”.

[27] Manfred Schmidt-Schau_, David Sabel Goethe-University, Frankfurt, Germany, ICFP '13, Boston, USA, Correctness of an STM Haskell Implementation.

[28] B. Liskov and R. Scheifler. Guardians and actions: Linguistic support for robust distributed programs. ACM Transactions on Programming Languages and Systems, 5(3):381–404, July 1983.

[29] J. Eliot B. Moss. Nested Transactions: An Approach to Reliable Distributed Computing.MIT Press, Cambridge, Massachusetts, 1985.

[30] Jeffrey L. Eppinger, Lily B. Mummert, and Alfred Z. Spector, editors. Camelot and Avalon: A Distributed Transaction Facility. Morgan Kaufmann, 1991.

[31] D. D. Detlefs, M. P. Herlihy, and J. M. Wing. Inheritance of synchronization and recovery in Avalon/C++. IEEE Computer, 21(12):57–69, December 1988.

[32] Nicholas Haines, Darrell Kindred, J. Gregory Morrisett, Scott M. Nettles, and Jeannette M. Wing. Composing first-class transactions. ACM Transactions on Programming Languages and Systems, 16(6):1719–1736, November 1994.

[33] Alex Garthwaite and Scott Nettles. Transactions for Java. In Malcolm P. Atkinson and Mick J. Jordan, editors, Proceedings of theFirst International Workshop on Persistenceand Java, pages 6–14. Sun Microsystems Laboratories Technical Report 96-58, November1996.

[34] Richard J. Lipton. Reduction: a new method of proving properties of systems of processes. InProceedings of the 2nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 78–86. ACM Press, 1975.

[35] Shaz Qadeer, Sriram K. Rajamani, and JakobRehof. Summarizing procedures in concurrent programs. In

Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principlesof programming languages, pages 245–255. ACM Press, 2004.

[36] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. Atomic Transactions. Morgan-Kaufmann, 1994.

[37] Panos Chrysanthis and Krithi Ramamritham. Synthesis of Extended Transaction Models Using ACTA. ACM Transactions on Database Systems, 19(3):450–491, 1994.

[38] Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques. Data Management Systems. Morgan Kaufmann, 1993.

[39] Andrew Black, Vincent Cremet, Rachid Guerraoui, and Martin Odersky. An Equational Theory for Transactions. Technical Report CSE 03-007, Department of Computer Science, OGI School of Science and Engineering, 2003.

[40] Tom Chothia and Dominic Duggan. Abstractions for Fault-Tolerant Computing. Technical Report 2003-3, Department of Computer Science, Stevens Institute of Technology, 2003.

[41] N. Busi, R. Gorrieri, and G. Zavattaro. On the Serializability of Transactions in Java Spaces. InConCoord 2001, International Workshop on Concurrency and Coordination, 2001.

[42] R. Bruni, C. Laneve, and U. Montanari. Orchestrating Transactions in the Join Calculus.In 13th International Conference on Concurrency Theory, 2002.

[43] E. Preston Carman, Jr.1, Till Westmann2 §, Vinayak R. Borkar3*, Michael J. Carey4, Vassilis J. Tsotras1*1University of California, Riverside 2Couchbase 3X15 Software, Inc. 4University of California, Irvine Email: ecarm002@ucr.edu* A Scalable Parallel XQuery Processor 2015 IEEE International Conference on Big Data (Big Data)978-1-4799-9926-2/15/$31.00 ©2015 IEEE 164.

[44] Shreya Banerjee and Anirban Sarkar “Ontology-driven approach towards domain-specific system design “.International journal semantics and ontologies, vol 11, no 1, pp- 39-60.

[45] ACID vs. BASE: The Shifting pH of Database Transaction Processing,By Charles Roe ,www.dataversity net.

Authors’ Profiles

Rita Ganguly, received the M.Tech degree from the NIT, Durgapur, India and entitled her name as a Research Scholar (Part-time) in Computer Science department(formerly known as Computer Application department.) ,NIT, Durgapur under the supervision of Dr. Anirban Sarkar. Presently she is working as an Asst. Prof of Computer Application Department, in Dr. B.C.Roy Engineering College, Durgapur, India.

Anirban Sarkar is presently a faculty member in the Department of Computer Applications, National Institute of Technology, Durgapur, India. He received his PhD degree from National Institute of Technology, Durgapur, India in 2010. His areas of research interests are Database Systems and Software Engineering. His total numbers of publications in various international platforms are above 100. He is actively involved in collaborative research with several Institutes in India and USA and has also served in the committees of several international conferences in the area of software engineering and computer applications.

How to cite this paper: Rita Ganguly, Anirban Sarkar, "An Approach to Develop a Transactional Calculus for Semi-Structured Database System", International Journal of Computer Network and Information Security(IJCNIS), Vol.11, No.9, pp.24-39, 2019.DOI: 10.5815/ijcnis.2019.09.04