Specifying Interdependent Data:  
A Case Study At Bellcore  

George Karabatis and Amit P. Sheth  
Bellcore  
444 Hoes Lane  
Piscataway NJ 08854-4182  

Abstract  
Poor specification of consistency causes related (target) data to be compromised. Lack of data consistency results in poor service because it hampers interoperability and results in wasted operations. It also incurs higher expense because it leads to errors requiring manual operations or system patches. Consistency of corporate data requires managing interdependent data which are logically related data maintained by multiple systems and their databases. Last year, we proposed a framework for specifying interdependent data. In this framework, the interdependency between multiple systems is represented using Data Dependency Descriptors ($D^3$). The framework has three components – data dependency to specify how are the data related, consistency requirements that must be satisfied by these related data items, and consistency restoration procedures that specify when and how should the consistency be enforced.

This memorandum reports the results of a study to understand applicability, adequacy and advantages of our framework. We studied several Bellcore systems that already exist or are in development, and identified examples of actual and realistic interdependencies among them. The examples demonstrate a variety of complexity that could arise when specifying the three components of the framework. This empirical study shows that the framework allowed precise and detailed specification of complex interdependencies. This can lead to appropriate, more sophisticated, and more efficient enforcement strategies and better support of business rules related to consistency of corporate data. We believe our specification framework can help both information system developers and users by facilitating development of products that meet a business’s data consistency needs, facilitate interoperability of application systems, reduce need of time consuming and costly manual operations, and provide better quality of data to end users. While this report is primarily addressed to practitioners, we believe researcher can benefit from our attempt to understand and portray complex real world needs.

1 Introduction  

Most large companies, including Bellcore and Bellcore Client Companies (BCCs)\(^1\) have many systems\(^2\) to serve their business needs. One of the significant problems in their multi-system environments is managing the consistency of inter-related (interdependent) data stored in the databases of multiple autonomous and heterogeneous systems.  

\(^1\)Bellcore Client Company (BCC), as used in this document, means any divested Bell Operating Company, or its successor, or any regional affiliate thereof.  

\(^2\)In this document a “system” is an existing application system with its own database(s), e.g., an Operation Support System, or a data layer building block in the OSCA\(^\text{TM} \) Architecture. OSCA is a trademark of Bellcore.
data for the end users (and may result in poor service to their customers), difficulty in supporting interoperability of various systems used by BCCs, and/or expensive manual operations to restore consistency of data. With the current emphasis on data as a corporate asset and its integrity [Mil89], the management of interdependent data is receiving more attention [SK89][RSC90][SI92].

Management of interdependent data is important to address the following business requirements:

- reducing currently labor intensive errors involving manual intervention (called requests for manual assists – RMAs), and ad-hoc system patches or work arounds related to data inconsistency.

- better serving the customers and reduce wasted operations by improving the quality of data by adhering to the business rules related to data consistency (e.g., through declarative specifications and selective update propagation).

- improving the requirements specification to better capture requirements for product revisions or new products, improving the value of Bellcore’s product by capturing information needed to interoperate among systems, and improving data layer building block (DLBB) developments by allowing better requirements exchange between product requirements analysts and DLBB developers.

In the majority of existing applications, the mutual consistency requirements among multiple databases are either ignored, or captured using the CRUD (Create, Read, Update, Delete) rules in the information model and embedded in the application code. While formal and precise languages appropriate for CRUD rule specification may exist [Kil91] (e.g., predicate calculus, Z, etc.), the practice up till now has been to use mostly textual descriptions to specify CRUD rules.

We believe that specification of interdependency with the following desirable characteristics can be of significant value in meeting the above specified business needs.

- Specification framework should be comprehensive, allowing representation of all aspects that have bearing on their specification and enforcement. It should allow capturing of business requirements affecting the requirements of desired levels of consistency of data.

- Specifications should be declarative rather than procedural. The separation of the constraints from application programs facilitates the maintenance of data constraints and allows flexibility in their implementation.

- Specification framework and languages used should facilitate evaluation of the correctness of specifications.

- Languages and interfaces should be appropriate for persons providing the specifications (e.g., requirements analysts) and developing an implementation to enforce them (e.g., building block developers).

- Specifications should facilitate their enforcement either by application programs or automatically by the system (i.e., in this context, specifically the data layer). The latter may involve using Polytransactions [SRK92] and the emerging multidatabase transaction technology.

A framework for specifying interdependent data, developed to address the above issues was proposed in [SR90, RSK91]. Alternative specifications, in most cases focusing on particular types

---

3A DLBB is an OSCA™ term specifying system components that manage data.
of interdependencies, appear in [WQ87, ABGM90, DBB+88, SI92, S92] (see [SRK92] for additional references and a discussion on some of the related work). Preliminary issues related to correctness and enforcement of interdependent data were also discussed in [SRK92] and [SLE91].

During the summer of 1991, we undertook an effort to empirically study some of Bellcore’s systems to derive real and realistic examples and represent them in the framework of [RSK91], get feedback from persons who could benefit from the use of this work and get an understanding of how well the proposed framework meets the above business requirements and desirable characteristics. The issues of enforcement of the specification were not investigated.

The findings of the above study are reported here. The study included many meetings, with product requirements analysts and system developers who provided information relevant to the study, and a consideration of ten systems. Most of the discussions involved systems from the Planning and Engineering area of Bellcore, and their interactions with the source or fundamental data systems (i.e., provisioning and unit inventory systems) and reference data systems. Figure 1 shows a high level view of the systems that featured in the discussions. Think arrows show the interdependencies that are documented in this memorandum. In many cases, many similar interdependencies of the same type existed between the systems. We did not document all interdependencies, but only those involving different types of specification. Thin arrows show interdependencies that were identified but not documented.

Section 2 reviews our specification framework proposed in an earlier memorandum. Section 3 provides a very brief introductions of the systems for which example interdependent data are documented. Section 4 is the main section of this memorandum and documents specifications of several interdependent data. Sections 5 and 6 discuss the findings and provides some conclusions. The appendix at the end gives a glossary of acronyms.

## 2 Brief Overview of the Interdependency Descriptor

In this section we present a brief overview of our work on a framework that can be used to specify interdatabase dependencies between data objects. More detailed description on the specification of interdependencies between related data can be found in [RSK91][SRK92]. A reader who is familiar with the above references may wish to skip this section.

Our framework for specifying interdependent data consists of two aspects: a formal model of interdatabase dependency specification called data dependency descriptor \( D^3 \) consisting of three components, and language(s) for providing detail specification of each component. As we discussed in [RSK91], it is the model that is important. There are many alternatives for the specific languages proposed in [RSK91] (e.g., see [SI92][Kil91]).

A \( D^3 \) is a 5-tuple:

\[
D^3 = \langle S, U, P, C, A \rangle
\]

where:

- \( S \) is the set of source data objects,
- \( U \) is the target data object,
- \( P \) is a boolean-valued predicate called interdatabase dependency predicate. It specifies a relationship between the source and target data objects, and evaluates to true if this relationship is satisfied.
- $C$ is a boolean-valued predicate, called *mutual consistency predicate*. It specifies consistency requirements and defines when $P$ must be satisfied.

- $A$ is a collection of *consistency restoration procedures*. Each procedure specifies actions that must be taken to restore consistency and to ensure that $P$ is satisfied.

$P$, $C$, and $A$ are the three components of $D^3$ (see Figure 2). Specification of $S$ and $U$ is for convenience and can be removed by adding directionality (from sources to target) in $P$.

The dependencies are specified in a declarative fashion and will be viewed as separate schema entities, collectively referred to as *interdependency schema (IDS)*. In other words, $IDS$ is a set of all $D^3$s to be enforced in the multidatabase system for managing consistency of the corporate data. They can be used to perform updates in a way that allows us to overcome the limitations imposed by traditional transaction management systems (e.g., two-phase commit). For example, we may allow data to be temporarily inconsistent, within specified limits.

\[\text{Interdependent Data}\]

\[\begin{array}{c}
\text{Dependency (P)} \\
\text{Consistency (C)} \\
\text{Restoration (A)}
\end{array}\]

\[\text{Temporral component} \quad \text{Data State component}\]

Figure 2: Major Components of a $D^3$

*The languages for specifying each component of a $D^3$ we have proposed are extensible.* The rest of the section provides a brief overview (for more detail, see [RSK91] [SRK92]).

### 2.1 Specification of the Dependency Predicate in $D^3$

Dependency predicates can be specified, using operators of relational algebra (selection, projection, join, union, difference, intersection, etc.). Together with the basic operators, we also use the aggregate operator $\xi$ [Klu82], and the transitive closure operator $a$ [Agr87]. The $\xi$ operator allows specification of aggregate functions such as *sum* or *count* for the whole relation or for groups obtained by partitioning the relation according to the specified attribute. The $a$ operator computes the transitive closure of a single relation $R$, assuming that the relation is transitive over its first two attributes. Aggregate and grouping operators can be used inside the $a$ operator. Depending on the user requirements and the data model supported by the DBMSs, alternative languages such as SQL (if necessary with some extensions) or a language based on an extended Entity-Relationship model may be used. In Section 4, we use extended relational algebra for all examples, but also show alternative SQL-like specifications for specifying dependency predicates of three of the examples.
2.2 Specification of Mutual Consistency Requirements in $D^3$

The consistency requirement predicate, denoted by $C$, specifies when the related data must be consistent. The predicate consists of two largely orthogonal components – time and state of data.

Interdependent data objects may be allowed to be inconsistent within certain limits, determined by $C$. The specification of the consistency predicate can involve multiple boolean valued conditions be referred to as consistency terms and denoted by $c_i$. A consistency term has a boolean value. Its value is false when the related data are within the allowable limit of inconsistency, and becomes true when the limit for allowable inconsistency is exceeded. Each $c_i$ refers to a mutual consistency requirement involving either time or the state of a data object. Hence, $C$ is a logical expression involving $\lor$, $\land$, and $\neg$ operators and consistency terms $c_i$.

2.2.1 Temporal Consistency Terms

The following consistency requirements involving time may be defined:

- At a particular date and/or at a specific point in time. For example, 
  @9:00 means at 9.00 o'clock, 
  on 27-May-1991 means 'on 27th of May, 1991'.

- The position of the ! is used to distinguish between before and after. For example, 
  ! 8:00 means before 8 a.m. 

- We can specify intervals of time or/and date using the $\Delta$ operator. $\Delta (9:00 - 17:00)$ means during the whole interval between 9 a.m. and 5 p.m. 
  $\Delta (10-Jun-1991@17:00 - 11-Jun-1991@8:00)$ specifies an overnight interval.

- Periodically, when certain amount of time has elapsed. We use the expression $\varepsilon(period@t)$, to specify 'every period of time at time t'. For example, 
  $\varepsilon(day @17:00)$ means 'every day at 5 p.m.'.

It is also possible to specify a time interval or period relative to a specific event. For example, a period relative to the last consistency restoration time (i.e., the time when dependency between the related data was enforced) is specified as $\varepsilon*(period@t)$.

2.2.2 Data State Consistency Terms

Consistency terms involving data values, can specify the following constraints:

- the maximum change in the value of data which is allowed on a data object
- a condition involving data values or an aggregate function on the values of source data items
- the maximum allowed discrepancy between version numbers of a given object that can exist in related databases
- a number of updates that can be performed on a given source object before the mutual consistency is restored with the target object
- a specific operation is performed before or after mutual consistency must be restored
2.3 Specification of consistency restoration procedures in $D^3$

As mentioned before, whenever the consistency predicate evaluates to true and the dependency predicate is not satisfied, inconsistencies between the source and target objects cannot be further tolerated. The action component $A$, specifies conditional execution of one or more consistency restoration procedures.

The syntax of the $A$ component allows the specification of either single or multiple consistency restoration procedures. $A$ with single consistency restoration procedure is specified as:

$$A : procedure\ name [as\ execution\ mode]$$

Since there may be several ways to restore consistency, more than one consistency restoration procedure can be specified in the action component $A$, if needed. In this case we use the following notation:

$$\text{when } C_{R_1} \text{ do } procedure\ name [as\ execution\ mode]$$
$$\vdots$$
$$\text{when } C_{R_n} \text{ do } procedure\ name [as\ execution\ mode]$$
$$\text{otherwise default procedure\ name [as execution mode]}$$

The execution mode of the action component in the $D^3$, specifies the way by which a restoration procedure (child) is related to its parent transaction. A child is coupled if the parent transaction must wait until the child transaction completes before proceeding further. It is decoupled if the parent transaction may schedule the execution of a child transaction and proceed without waiting for the child transaction to complete.

If the dependency schema requires immediate consistency, the nested transaction model may be used, in which the descendent transactions are treated as subtransactions which must complete before the parent transaction can commit. A coupled transaction can be vital in which case the parent transaction must fail if the child fails, or non-vital in which case the parent transaction may survive the failure of a child [G+90].

3 Overview of the systems studied

In this section, we give a brief introduction to the systems for which we give examples of interdependent data using $D^3$s in the next section.

Planning View of Inventory (PVI)

A plan utilizes information stored in the inventory databases. The portion of inventory relevant to planning is stored in the PVI DLIB. PVI supports integrated planning and traffic engineering functions within the Bellcore Client Companies (BCCs). PVI data are critical in supporting the INPLANS applications (such as INPLANS/ITP and INPLANS/PTC) and in development of current and future planning systems.

PVI contains data that are derived from or are closely related to many other corporate data repositories of strategic importance. Some of these related corporate data are managed by various databases and systems, such as the Unit Inventory (UI), Loop Facilities Assignment and Control System (LFACS), TIRKS®, et c. From the PVI viewpoint, these data are called fundamental data.

---

TIRKS is a registered trademark of Bellcore.
Existing planning systems such as \textit{LEIS$^TM$/PLAN}, Network Planning System Facility (NPS-F), Network Planning System Traffic (NPS-T), Facility Equipment Planning Systems (FEPS), \textit{SEAS$^TM$/and Local Switching Demand & Facility (LSD&F)} contain views of fundamental data. In many cases, PVI and other databases or DLBBs (e.g., Demand Forecast Group (DFG) and Sub-Assembly Model (SAM)) accommodate further views of the data in existing planning systems. In some cases in future, the existing planning systems will give up the data they manage to PVI DLBB or other DLBB, and turn into process layer building blocks (PLBBs) that use a DLBB.

Data in PVI that we dealt with represent summary or aggregation of quantities of equipment and facilities stored at locations of various granularity. In the next section, we document two examples of interdependent data involving LEIS/PLAN and PVI (see $D_3^3$ and $D_3^3$), and two involving LSD&F and PVI (see $D_3^3$ and $D_3^3$).

**Planning Systems**

LEIS/PLAN is a Bellcore Planning System that contains aggregated information about carrier system slot assignments. The data in LEIS/PLAN represent summary information of fundamental data in LFACS. Some of the aggregated information of LEIS/PLAN includes counts of spare equipment per geographic area. These can be found in the CSA\_SPARE table of LEIS.

LSD&F is a Bellcore system that is responsible for managing the switch data DLBB. These data represent detailed information about the capacity in various network access lines and are used by INPLAN applications that require \textit{switch data} residing in corporate DLBBs.

**Engineering Systems**

A job is a formally stated description of work to be done and, optionally, material to be placed. Examples of traditional engineering job include placing cables and central office switches. The Job database is the central repository for common information concerning jobs for BCCs. The Job database, along with other databases such as Unit Inventory, Location, and Catalog, provides data to support the Engineering Design and Implementation Management tasks. In the next section, we give two examples of interdependent data. $D_3^3$ involves Job Object with Company Organization entity in the Organization database. $D_3^3$ involves Job entity and Field Reporting Code entity Financial Accounting Group databases.

**4 Specific $D_3^3$s**

In this section, we specify six $D_3^3$s. The $D_3^3$s discussed are based on actual, intended or desirable dependency among data. However, no guarantee is made that they are currently supported or will be supported in future. The relational table descriptions are based on the logical data model descriptions. In most cases, only partial schema of the tables that are relevant to the examples are given, and existing or future database implementation may not use the same table schemas.

The first two $D_3^3$s are specified between the interdependent data in LEIS and PVI.

\begin{itemize}
\item LEIS is a trademark of Bellcore.
\item SEAS is a trademark of Bellcore.
\end{itemize}
The relation $DMD\_CAP$ in PVI contains various counts of demand and capacity for an instance of Planning Unit. The attribute $spare$ in table $DMD\_CAP$ contains the spare capacity of an inventory item, based on some measurement unit. This information is also kept in LEIS, in table $CSA\_SPARE$ under the $pots\_spr\_npl$ attribute. The $D^3$ specifies the consistency requirement (discussed in detail below) between the spare capacity of an inventory item in LEIS and the spare capacity of the same item in PVI.

In all the relation schemas presented below, the underlined attributes represent keys. The relevant schema of $PVI.DMD\_CAP$ is:

$DMD\_CAP$ ($pln\_unit\_id$, $dmd/cap\_meas\_unit$, $load\_time\_period$, $location\_id$, $total$, $assigned$, $spare$, $tradeable$)

$LEIS.CSA\_SPARE$ is defined as:

$LEIS.CSA\_SPARE$ ($csa$, $csa\_flag$, $dss\_flag$, $dss\_spr\_pl$, $pots\_spr\_npl$, $avl\_pg\_cap$, $spr\_fibers$)

The attribute $pots\_spr\_npl$ in table $CSA\_SPARE$ is the source object which must equal the attribute $spare$ in table $DMD\_CAP$ of PVI, under the conditions $pln\_unit\_type$ = 'all dlc sys' and $dmd/cap\_meas\_unit$ = 'npl channel'. For these related objects, we can allow the target object to diverge with respect to the source object (a) by up to 6 months (defined by $c_1$), or (b) until a planner explicitly requests an Inquiry of the most recent data for a particular carrier service area ($csa$) (defined by $c_2$). In usual circumstances, a restoration procedure $Update\_PVI.DMD\_CAP$ will be executed once every six months to restore the dependency as required by $c_1$. It will involve data for all $csa$ the LEIS system maintains. Today, this restoration involves a bulk upload. When a planner wants most recent data for a specific $csa$ of interest that is supported by $c_1$ before s/he begins the planning activity, s/he issues the Planning\_Inquiry. This is particularly useful when a planner knows or suspects that a particular $csa$ has seen much higher than usual recent provisioning activity. Procedure $Update\_PVI.DMD\_CAP\_{csa\_Only}(csa)$ will involve consistency restoration for interdependent data extensions related to the $csa$ specified in the $Planner\_Inquiry(csa)$ only. This restoration should be done in a coupled and vital mode since the planning activity can proceed only after its successful execution.
\[ D_3 \]
\[ S : LEIS.CSA.SPARE.pots_spr_npl \]
\[ U : PV1.DMD.CAP.spare \]
\[ P : \Pi_{casa,pots_spr_npl}( LEIS.CSA.SPARE) = \]
\[ \Pi_{location_id, spare \mid pln_unit_type = 'all dle sys' \land dmd/cap_meas_unit = 'npl channel'}( PV1.DMD.CAP) \]
\[ C : c_1 \lor c_2 \text{ where} \]
\[ c_1 := \varepsilon(6 \text{ months}) \]
\[ c_2 := \text{Planner\textunderscore Inquiry}(casa) \]
\[ A : \text{when } c_1 \text{ Update\textunderscore PV1.DMD.CAP as non\textunderscore coupled} \]
\[ \text{otherwise Update\textunderscore PV1.DMD.csaOnly(csa) as coupled and vital} \]

The above \( P \) component can be specified as \( P_S = P_U \), where \( P_S \) and \( P_U \) can be expressed in SQL-like syntax as follows:

\[ P_S \text{ is} \]
\[ \text{SELECT csa, pots_spr_npl} \]
\[ \text{FROM LEIS.CSA.SPARE;} \]
\[ \text{and } P_U \text{ is} \]
\[ \text{SELECT location_id, spare} \]
\[ \text{FROM PV1.DMD.CAP} \]
\[ \text{WHERE pln_unit_type = 'all dle sys' AND} \]
\[ \"dmd/cap_meas_unit\" = 'npl channel'; \]

It is interesting to note in this example that \( c_1 \) represents what is done today, while \( c_2 \) represents what is also desirable but is not supported today. As the technology evolves or the system is implemented in stages, a system implementer can improve enforcement of the specifications leading to better quality of data available to the users.

\[ D_2 \]

Our second \( D_2 \) is used to describe a different relationship between database objects of the same tables used in the previous \( D_3 \). More specifically we need to describe a relationship between attribute \textit{spr\_fibers} from \textit{LEIS.CSA.SPARE} and attribute \textit{spar}e from \textit{PV1.DMD.CAP} table. This relationship identifies that number of spare fibers in a particular \textit{csa} in LEIS, should be equal to the value of \textit{spar}e in \textit{PV1.DMD.CAP}, under the conditions that \textit{pln\_unit\_type} = 'e/o conv' and \textit{dmd/cap\_meas\_unit} = 'fiber'. The values on these two attributes can be inconsistent up to a period of six months (consistency term \( c_1 \)). Currently, inconsistencies of this nature are restored with the use of bulk uploads every six months. We also specify that whenever a planner requests an Inquiry for a particular \textit{csa}, probable inconsistencies must be also eliminated (consistency term \( c_2 \)). Restoration of violated consistency can be achieved by the execution of the restoration procedures. The choice of the procedure to be executed is decided by the consistency terms \( c_i \).

<table>
<thead>
<tr>
<th>LEIS.CSA.SPARE</th>
<th>PV1.DMD.CAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>csa, csa_flag, dss_flag, dss_spr_pl</td>
<td>pln_unit_id, dmd/cap_meas_unit,</td>
</tr>
<tr>
<td>pots_spr_npl, avl_pg_cap,</td>
<td>pln_unit_type, location_id,</td>
</tr>
<tr>
<td>spr_fibers</td>
<td>total, assigned, spare, tradeable</td>
</tr>
</tbody>
</table>
There are a number of cases in the planning system where one system contains an itemized and detailed description of an inventory unit and another system contains only the total number of these inventory units of the same type. One such case occurs between table LSD&F.ENTITY_JOB and PVI.DMD_CAP.

LSD&F.ENTITY_JOB is defined as:

ENTITY_JOB (entity, entity_cut, ent_in_cap, eq_type, job_num, incap_in_nal, incap_in_ins)

LSD&F contains detailed information about line capacity in network access lines (incap_in_nal) for every equipment connected to a particular switch, identified by its CLLI-code. On the other hand, PVI contains a total of the line capacity for all switches. The dependency constraint identified above may not hold after an update to the LSD&F database, such as a recent job that connected more cable lines to a switch with a specific CLLI-code, e.g., 12345678. Still PVI can tolerate such updates in LSD&F up to a point specified in the consistency predicate C before it must be made consistent with the related data in LSD&F. Assuming that LSD&F keeps versions of the ENTITY_JOB table, PVI can lag behind, up to 3 versions. In addition, if 15% or more of the tuples of the particular switch in ENTITY_JOB have been updated, then the consistency should be restored immediately. Currently, there is no provision to maintain this kind of interdependency between the above described data items. The current solution involved bulk uploads from the LSD&F into PVI, usually once every six months. We can formulate the above interdependency using the $D_3^3$ below:

$$D_3^3::$$

$$S: LEIS.CSA.SPARE.spr.fibers$$

$$U: PVI.DMD_CAP.spare$$

$$P: \Pi_{csa.spr.fibers}(LEIS.CSA.SPARE) =$$

$$\Pi_{location_id, spare}(PVI.DMD_CAP)$$

$$C: c_1 \land c_2$$ where

$$c_1 := \varepsilon(6\ months)\ and$$

$$c_2 := \neg\text{Planner.Inquiry}(csa)$$

$$A: \text{when } c_1 \text{ Update } PVI.DMD.CAP \text{ as non-coupled}$$

$$\text{otherwise Update } PVI.DMD.csaOnly(\text{csa}) \text{ as coupled and vital}$$

The above P component can be specified as $P_S = P_U$ where $P_S$ and $P_U$ can be specified in SQL-like syntax as follows:

$P_S$ is

SELECT csa, spr.fibers
FROM LEIS.CSA.SPARE;

and $P_U$ is

SELECT location_id, spare
FROM PVI.DMD_CAP
WHERE plan_unit_type = 'c/o conv' AND
      "dmd/cap_meas_unit" = 'fiber';
In the next $D_3$ we have two source data objects, namely two attributes from two different tables of the LSD&F database. One of the relevant tables is ENTITY JOB that was defined above. The other table is LSD&F: YEAR, defined as:

$$LSD&F: YEAR = (\text{nal}_dmd, \text{ln}_dmd, \text{ces}_dmd, \text{orig}_dmd, \text{term}_dmd)$$

The target data object is the attribute assigned in PVI.DMD_CAP. We need to specify that the demand on a network access line of a specific switch in LSD&F must match the value of the assigned attribute in the PVI for the same switch if capacity unit = 'nal'. We can tolerate inconsistencies between related type to a specified extend: either up to the point that the demand on a network access line (nal_dmd) represents at least 90% of the line capacity (consistency term $c_1$) or whenever there are less than 5000 free lines remaining (consistency term $c_2$). The two consistency terms are linked with different actions that will restore consistency. In case of $c_1$, we invoke a regular Planning non-coupled procedure. But if $c_2$ occurs, the number of free lines are below the minimum and an Emergency Planning procedure must be executed. Notice the difference on the coupling modes of the above two procedures. The Emergency Planning procedure has to be executed in the coupled and vital mode since we do not want application using inconsistent PVI data to continue before the data is made consistent.
\[
D_4^3 ::
S : \{ \text{LSD} \& F. \text{YEAR.nal.dmd}, \text{LSD} \& F. \text{ENTITY.JOB.lncap.in.nal} \}
\]
\[
U : \text{PV1.DMD_CAP.assigned}
\]
\[
P : \Pi_{\text{DMD.DMD_CAP.assigned}(\text{DM D.DMD_CAP.pln_unit_id, dmd/cap_meas_unit, pln_unit_type, location_id, total, assigned, spare, tradeable})} =
\xi_{\sum\text{dmd/ent.long.in_cap, entity_unit_id, pln_unit_id}}(\text{DM D.DMD_CAP}) =
\xi_{\text{Sum[nal.dmd, entity.cut, job_num, ...]}(\text{YEAR \& ENTITY.JOB})}
\]
\[
C : c_1 \lor c_2 \text{ where}
\]
\[
c_1 := (\text{YEAR.nal.dmd/ENTITY.JOB.lncap.in.nal}) \geq 0.9
\]
\[
c_2 := (\text{ENTITY.JOB.lncap.in.nal - YEAR.nal.dmd}) < 5000
\]
\[
A : \text{when } c_1 \text{ Planning as non-coupled}
\]
\[
\text{when } c_2 \text{ Emergency Planning as coupled and vital}
\]

\[
D_5^3
\]

The following \(D_5^3\) represents a case of specifying a referential integrity constraint. It is one of the most frequently occurring type of interdependent data. In this particular example we want to specify the following existential constraint: every organization inside the \text{JOB} database must exist in the \text{ORG} database. Violation of this consistency constraint cannot be tolerated, hence the \text{immediately} consistency constraint. Because no inconsistency is allowed, no automatic consistency restoration procedure is specified. Instead, the update is disallowed (aborted) and the user is notified.

\[
\begin{array}{|c|}
\hline
\text{JOB.JOB\_ORGANIZATION} & \text{ORG\_COMPANY\_ORGANIZATION} \\
\text{Organization Id, ...} & \text{Organization Id, ...} \\
\hline
\end{array}
\]

\[
D_6^3 ::
S : \text{JOB.JOB\_ORGANIZATION.Organization Id}
\]
\[
U : \text{ORG\_COMPANY\_ORGANIZATION.Organization Id}
\]
\[
P : \Pi_{\text{JOB\_ORGANIZATION\_ORGANIZATION.Organization Id} \subset \Pi_{\text{COMPANY\_ORGANIZATION}}(\text{COMPANY\_ORGANIZATION})}
\]
\[
C : \text{immediately}
\]
\[
A : \text{abort update on JOB\_ORGANIZATION and Notify User}
\]

\[
D_6^3
\]

Here we are going to describe another situation of dependency between the previously defined \text{FIN\_ACC\_GROUP} and \text{JOB} databases. The \text{FIN\_ACC\_GROUP\_FRC} table among other things, contains information about forecast dollars for every job. The relevant schema of this table is:
\text{FIN\_ACC\_GROUP\_FRC} (\text{frc, description, job_no, actual\$, forecast\$, actual_hours, forecast_hours})
The table JOB from the JOB database also contains some information about the forecast dollars required to complete the job. The relevant schema of this table is:

\[ \text{JOB.JOB (job_no, budget_id, frc, actual$, forecast$, viertype, actual_hours, forecast_hours).} \]

We want to capture semantic information relating the forecast dollars in each table. The following \( D^3 \) specifies that the forecast dollars in the JOB table for a particular job, should be less or equal to the forecast dollars in the FRC table for the same job, which is easily captured with the \( P \) predicate. The Financial Accounting Group in this case uses a quarterly cycle. Updates to forecast$ that reflect changes in dollars for cases when viertype is neither staff nor dollar have to be propagated immediately. This is specified in term \( c_1 \). However, if the value of the JOB.viertype attribute is staff or dollar then we can keep the job going with inconsistent dollar data as follows: if we are currently in the first ten week period of a quarter (defined by value \( true \) for a boolean value variable \( p \) representing the first ten week period of a quarter), the job need not be updated for up to two weeks (consistency term \( c_2 \)). But, if we are in the last two week of a quarter (identified by \( \neg p \)), then this update must be reflected before 8 a.m. of the next day (consistency term \( c_3 \)).

\[
\begin{array}{|c|c|}
\hline
\text{FIN.ACC.GROUP.FRC} & \text{JOB.JOB} \\
\hline
\text{frc, description, job_no, actual$, forecast$, actual_hours, forecast_hours} & \text{job_no, budget_id, frc, actual$, forecast$, viertype, actual_hours, forecast_hours} \\
\hline
\end{array}
\]

\[ D^3_6 :: \]
\[ S :: \text{FIN.ACC.GROUP.FRC.forecast$} \]
\[ U :: \text{JOB.JOB.forecast$} \]
\[ P :: \Pi_{\text{forecast$}} \sigma_{\text{job_no}=xyz} (\text{JOB}) \leq \Pi_{\text{forecast$}} \sigma_{\text{job_no}=xyz} (\text{FRC}) \]
\[ C :: c_1 \vee c_2 \vee c_3 \text{ where } \]
\[ c_1 :: \Pi_{\text{job_no}} \sigma_{\text{job_no}=xyz} (\text{FRC}) = \]
\[ \Pi_{\text{job_no}} \sigma_{\text{job_no}=xyz\&\text{viertype}=\text{staff} \& \text{viertype}=\text{dollar}} (\text{JOB}) \]
\[ c_2 :: \neg c_1 \land p \land t + 14 \text{days} \]
\[ c_3 :: \neg c_1 \land \neg p \land 8 \text{am} + 1 \text{day} \]
\[ A :: \text{Report to Accounting} \]

5 Discussion

In this section we evaluate the adequacy of the \( D^3 \) framework for specifying data interdependencies, and we describe the business advantages of using such a framework. Let us first comment on the criteria for a framework for specifying data interdependencies.

Comprehensive specification

With the \( D^3 \) framework we were able to model all business requirements related to data consistency which came up in our case study. We were able to represent all aspects of data interdependencies which have a bearing on specifying and enforcing them. Several \( D^3 \)s exhibit complex dependencies (\( P \)). \( D^3_2 \) and \( D^3_4 \) involve aggregation, and the latter also involving two source objects. Consistency requirements ranged from simple (e.g., \( D^3_1, D^3_2, \) or \( D^3_3 \)) to complex (e.g., \( D^3_6 \)).
Declarative and implementation independent specification

The $D^3$ framework satisfies this requirement. The specifications are declarative and they are completely independent of implementation platform, language, and application code.

Support evaluation of correctness

In general, correctness of complex specifications is not a well understood issue (see [SRK92] for further discussion). We have not yet attempted to investigate proofs of completeness, minimality and safety properties of the $D^3$ specifications. However, a necessary first step toward proof of correctness is a precise specification, and the $D^3$ framework supports this.

Easy to understand and appropriate for use

The users of data interdependency specifications are strategic planners, requirements analysts, DLBB developers, developers of other building blocks (e.g., PLBBs and ULBBs) which access DLBBs, and other users (e.g., clients’ systems analysts working on interoperability of Bellcore developed DLBB with other systems). The $D^3$ specifications in this study used syntax based on extended relational algebra for specifying dependencies. This choice was targeted toward the experienced developers. The analysts and developers with whom we worked on the case study experienced some unfamiliarity with the notation but found them understandable and reasonably usable. It is more appropriate for developers than for requirements analysts. SQL syntax demonstrated for the first three examples might be an even more usable alternative, although standard SQL may need to be extended in some cases. In the context of current information modeling practice, it would be desirable to specify $D^3$s at a higher level of abstraction, possibly with graphical notations. In fact, the requirements analysts with whom we worked expressed their desire for a comprehensive information modeling tool which includes support for specification of data interdependencies (e.g., the information modeling guidelines [INF91] that also allows them to specify interdependencies). The software prototype system, Bellcore Schema Design and Integration Toolkit (BERDI) [SM92], has demonstrated a first step in this direction by allowing specification of unstructured (textual descriptions) interdependencies between objects in different schemas.

Support enforcement

We did not consider enforcement issues in this study. However, in related work we are investigating a mechanism called polytransactions [SRK92] for automatic enforcement of consistency requirements. We believe that $D^3$s offer many desirable characteristics that CRUD rules do not currently provide.

Meeting Business Requirements

Figure 3 shows the potential beneficiaries of improved management of interdependent data and the benefits which each can potentially gain.
Capture business requirements about data quality/consistency, and interoperability

Better quality of data, knowledge of the consistency of data, fewer manual fixes

Figure 3: Beneficiaries of Improved Management of Interdependent Data

From a corporate-wide standpoint, an adequate framework for specifying data interdependency supports accurate modeling of requirements, which in turn leads to improved quality of data, which in turn reduces the need for rework and manual intervention during operations. This results in lower operations costs and faster and accurate response for customers. Accurate modeling of clients' requirements also improve the value of Bellcore's product by capturing information clients need to interoperate their systems with our products, and improve DLBB development by allowing better requirements exchange between requirements analysts and developers. A specification framework such as the one used here provides a vehicle for product requirements analysts to communicate with the clients and with the developers (and vice-versa). Figure 4 summarizes these advantages.

Figure 4: Advantages of Managing Interdependent Data
6 Conclusions

We have proposed a framework to support specification of interdependent data between autonomous, heterogeneous systems supported by Bellcore. We conducted a study in which many people from various backgrounds including requirements analysts, application developers and DLBB developers, were involved. This study required the identification of the interdependencies between data from different systems, which most of the time led us to extract the semantic meaning not only of relevant tables but also of attributes between systems. During the process of identifying existing interdependencies, we also encountered dependencies between data that are not supported so far, but could be of benefit if enforced. The case study allowed us to observe that the specification framework is comprehensive, application independent, and declarative. While it seems to facilitate evaluation of correctness of specification and enforcement, additional work is needed. We also observed characteristics of languages that can be used to specify the components of $D^3$s. Finally, we were able to observe that using the specification framework for managing interdependent data can lead to improved interoperability and quality of data, lower cost and faster response. We feel that this study on the applicability, adequacy and the advantages of the framework provides concrete and detailed information for further work leading to engineering and deployment activities to support consistency of corporate data. Our current research attempts to address the issues of correctness of specifications and execution of polytransactions for automatic enforcement of $D^3$s declared to the system.

Acknowledgements

This study would not have been possible without significant support from Mike Barlow, David Levin, Ed Yuen and Brian Nordeen, who helped us understand the systems (applications, databases and DLBBs) and devoted their time to support our "knowledge engineering" task, in identifying interdependent data examples. We would also like to thank Gomer Thomas, Frank Jacobsen, Dan Danning, Ashok Ingle, Tony Kwong, George Mitwasi, Hassan Srinidhi, Chris Whiting and Cheong Youn for their discussions. We acknowledge active collaboration with Marek Rusinkiewicz (Univ. of Houston) and the Carnot project (MCC, Austin) on the Management of Interdependent Data project.

References


Appendix

Glossary

CID  Catalog Item Database
CSA  Carrier Serving Area
DFG  Demand Forecast Group
DLBB Data Layer Building Block
FEPS  Facility Equipment Planning Systems
INFORMS Integrated Forecast Management System
INPLANS Integrated Planning Systems
ITP  Integrated Technology Planning
LERG  Local Exchange Routing Guide
LFACS  Loop Facilities Assignment and Control System
LOC  Locations
LSD&F  Local Switching Demand & Facility
NPS-F  Network Planning System Facility
NPS-T  Network Planning System Traffic
PGG  Product Group Guidelines
PVI  Planning View of Inventory
SAM  Sub-Assembly Model
UI  Unit Inventory