FROM A DATA DESCRIPTION POINT OF VIEW

Diane C. Pirog Smith
Division of Computer Science
University of Utah

Introduction

As an area closely involved in the manipulation of large masses of data, Information Retrieval should benefit from any research developments involving data storage and data access methods. Current research on generalized data description languages has lead to several interesting developments in the area of data management. In this paper we discuss these developments and show that Information Retrieval is indeed likely to benefit as a consequence.

Definition of Data Description

"Data description" is a term enjoying several distinct interpretations in the Programming Language and Information Retrieval areas. These different interpretations are caused by several factors. From a practical standpoint, they are caused by the application envisaged for the data description. While, from a technical standpoint, they are caused by the aspects of data being described and the level of detail to which the description is taken.

To appreciate these various factors, we begin by considering data as a linguistic entity. The data on a disk pack is a string of symbols (in binary code) embedded in a three-dimensional space. This string has a certain syntactic structure (particular symbol sequences are recognized as fields, records, etc.), and it has a certain semantic structure (particular symbols are interpreted as pointers, as delimiters, or as data items of various types). A description of the syntactic structure alone is usually sufficient to retrieve information from a file, but only by an exhaustive search through the symbol string attempting to match keys. However, if a description of the semantic structure is available, then retrieval becomes more efficient as the access paths embedded in the file can be utilized. A syntactic description then indicates how the format of the file is comprised of its component symbol strings, whereas a semantic description specifies how the access paths embedded in the file can be used to effect efficient storage and retrieval. One factor in a data description is that of syntactic versus semantic description.

A second factor relates to the level at which a file is treated. Users of computer systems are rarely concerned with controlling directly the processors invoked by their jobs. Programming aids such as compilers, operating systems and data base management systems are normally delegated the tasks of managing the details of their programs and data. These various processors contribute a hierarchy of successively refined levels to the syntax and semantics of data. Many attempts have been made to formalize these levels by decomposing the characteristics of stored data into meaningful partitions [Olle, Smith, Taylor, SDDTTG, Altman]. Though differing in the numbers of sublevels and distribution of characteristics, these attempts recognize the following levels of data syntax and semantics:

1) the Entity Level: this level is the representation of the user's view of his data. It consists of entities and relations over entities.

2) the Logical Access Level: this level consists of the characteristics of the access paths which implement the relations seen at the entity level.

3) the Physical Access Level: this level consists of the characteristics relating to how data is positioned on physical storage media.

A third and final factor in data description is the type of description itself. A description may be implicit in a program which creates and maintains a data structure or explicitly stated in a data description language in such terms as "item length" or "ordering."

In terms of these factors we can define a generalized data description language (gddl) as a language in which the syntax and semantics of stored data can be explicitly described at all levels of its implementation.

Now let us look at how these factors are reflected in practical data description languages (ddls).

Examples of Data Description Languages

Although the idea of a generalized data description language is relatively new, data description facilities came into existence with the first computers, as they are implicit in the order codes that position data and control storage media. Higher level programming languages like COBOL contain statements that describe explicitly particular levels of a data structure. Macro calls on operating system data management components imply complex logical and physical access structures such as indexed sequential and sequential access structures for tape and disk.

Current ideas about generalized and explicit data description languages have arisen from the development of generalized data base management systems (GDBMSs) and extensible languages.

Because GDBMSs attempt to provide their users with a variety of of structures, they require a language in which the users can describe the options they select. The more generalized the DBMS is, the more generalized its ddl must be. A proliferation of GDBMSs and ddls led CODASYL in the late 1960's to form a task group. Its purpose was to codify a ddl which could be used as a standard for GDBMS development [CODASYL].

Extensible languages seek to increase the choice of structures available to users of higher level programming languages. So again a language is required in which a user can describe the new data structures he wishes implemented.

The explicit ddls developed by the CODASYL Task Group and by extensible language researchers (e.g. [Standish]) are not generalized in the sense described above. The CODASYL ddl is concerned with the entity and logical access levels only. For example, while pointers may be specified by a user as a means of implementing access paths, their format, storage media implementation and access strategies are predetermined by the GDBMS and not by the user. The data description elements of extensible languages are concerned with specifying logical access structures and methods. Storage level considerations are predetermined by the language processor.

In 1972 the first versions of a generalized ddl were implemented at the Universities of Pennsylvania and Michigan [Rameriz, Fry]. These implementations will be covered in greater detail in later sections of the paper. Now we will examine some of the current and potential applications of ddls.

Applications of Generalized DDLs

The lack of standardization in storage formats and the consequent difficulty in sharing stored data have suggested the use of a standardized DDL as a documentation language. We envisage that in the not too distant future every stored file will be accompanied by a tag (probably machine readable) that describes the structure of the stored data. The machine readable ddl will be used to drive a generalized read routine, so that files will not become obsolete after conversions to new computer systems. We further foresee a machine readable ddl driving a generalized write routine for storing data onto different storage media. Such a processor will be used to create on demand new logical and physical access structures for any given set of data.

A processor combining the generalized description driven read and write routines forms a description-driven data translator that converts stored data from one structure to another.

Data Description-Driven Data Translation

The idea of description driven generalized data translation is clearly appealing and is receiving an increasing amount of attention. Theoretical techniques have been developed in [Smith 1971, Taylor and Fry] and the current thrust is to demonstrate practical feasibility. To this end the projects at the Universities of Pennsylvania and Michigan and the work of the CODASYL Task Group on Stored Data Description and Translation [Fry and Smith 1972] are noteworthy.

A translation processor accepts as input three descriptions and the set of data to be translated. The first description characterizes the logical and physical structure of the data to be translated (the source files). The second description characterizes the structure into which the source files are to be translated (the target structure). The third description identifies, for each element in the target structure, its source of values from the source files. The relationships between descriptions, data and processor are illustrated in Figure 1.

Present versions of generalized translators operate, in effect, "off-line." That is, they cannot produce translated data on demand in real-time. Their mode of operation is either "one-shot" or "multiple-shot." A one-shot translation task is one in which translation between source and target structures need be performed only once. Translations

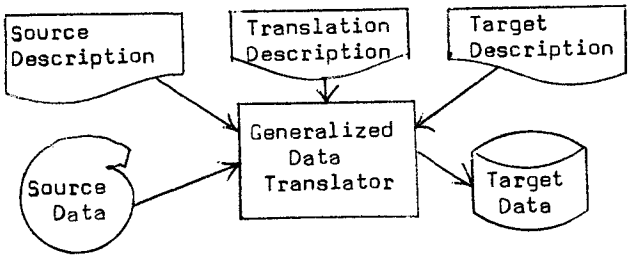

Figure 1. Relationships between a generalized data translator, descriptions and data.

necessitated by changes in computer systems are typical examples. The University of Michigan translator, as we will see later, is more oriented towards one-shot applications. A multiple-shot translation task is one in which translations between given source and target structures must be performed repeatedly. Sharing data between two separately maintained data bases is a situation which requires repeated translations to maintain updated versions of the respective bases. The University of Pennsylvania translator is oriented towards multiple-shot applications. These different modes of operation will be discussed in later sections.

While generalized data translation is proving to be a practical tool for off-line applications, there will be even more impressive applications should the technology advance to the state where real-time translation is feasible. Real-time translators could be incor-

porated into heterogeneous networks of computers to facilitate dynamic data sharing. They could be used with integrated data bases to allow users to impose multiple structures over single copies of data by dynamically translating between storage structure and any one of the users' desired structures.

With these current and future applications in mind we will now look at components and implementation strategies of data translators.

Components of a Generalized Data Translator

A generalized data description driven data translator consists of three components: a data structure analyzer (DSA), a data structure synthesizer (DSS) and a translation controller (TC).

The function of the DSA is to access items of data in a source file, given the description of the source file, the source file and a FIND command that identifies the item to be accessed. The DSA outputs the accessed item and returns control to the DSS. The relationships between the DSA, descriptions, data and control flow are illustrated in Figure 2.

The function of the DSS is to construct a representation of a target file, given the description of the target structure. It outputs a GET request to the TC to obtain the value for each element in the target structure it is constructing. When the value is found (by the DSA) it is sent as input to the DSS. Whatever conversion is

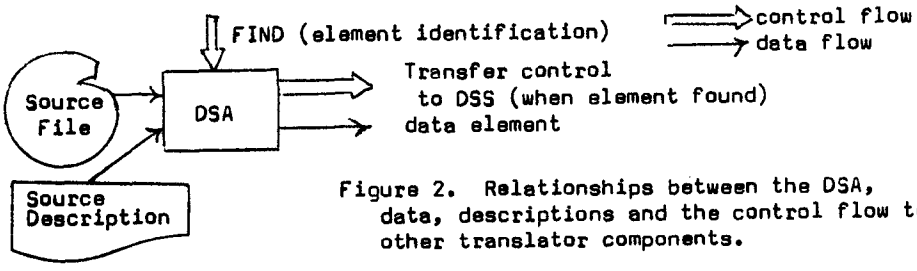

Figure 2. Relationships between the DSA, data, descriptions and the control flow to other translator components.

necessary is performed on the value before it is inserted in the target structure under construction. When the target structure is completed, it is output. The relationships between the DSS, data and descriptions and the DSS's control paths are illustrated in Figure 3.

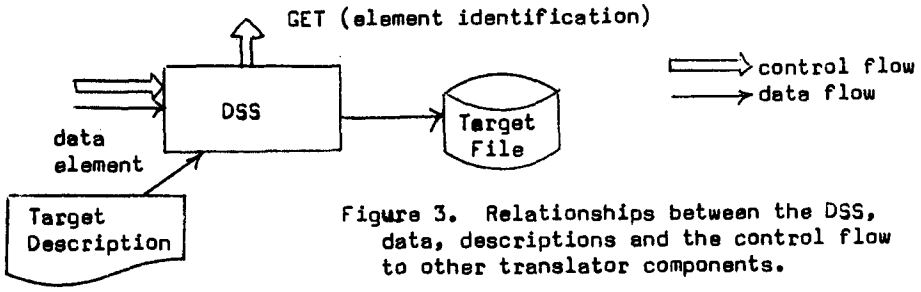

Figure 3. Relationships between the DSS, data, descriptions and the control flow to other translator components.

The TC is driven by a translation description which specifies what elements in a source file are to provide values for an element in the target file. It operates as follows: When it receives a GET command issued by the DSS , the element identification (ID) determines which target element is to be constructed next. The translation description is then consulted to yield the identification of the source element which is to provide its value. The TC outputs an appropriate FIND command to the DSA.

The generalized description-driven translator is produced by connecting the DSA, DSS and TC as illustrated in Figure 4. The pro-

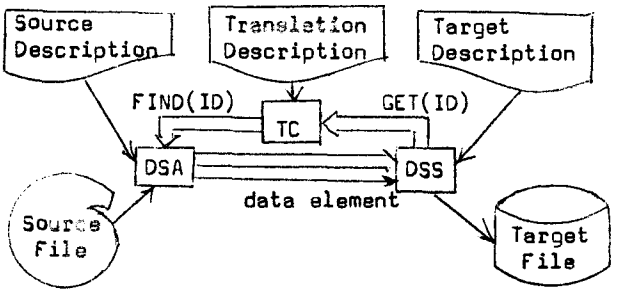

Figure 4. The structure of a data translator

cess begins with the DSS requiring a value for a data element to be supplied by the source file. The request is sent to the TC which determines the identification of the source element which is to provide the value. This identification is sent to the DSA in a FIND command. The DSA then returns control and the value to the DSS when it has extracted the value from the source file.

Data Translation Implementation Strategies

In developing implementations of a generalized data translator two different strategies of note have appeared. The first might be called the "Generalized Processor" approach and the second the "Compiler-compiler" approach.

The Generalized Processor approach is the strategy used at the University of Michigan. It is oriented towards use with generalized data base management systems. That is, the input to the translator will be records output by DBMSs in linear fashion. Its output will be records in formats that can be input by DBMSs. Later versions will accept as input files created by DBMSs and will output files directly usable by DBMSs. The translator is implemented as three separate modules: two "convertors" and a restructurer. The convertors are generalized read and write routines that convert data into and from a normalized form. This normalized form is a standard representation for all access paths and value encodings that is machine independent. The components are linked as illustrated in Figure 5. The restruc-

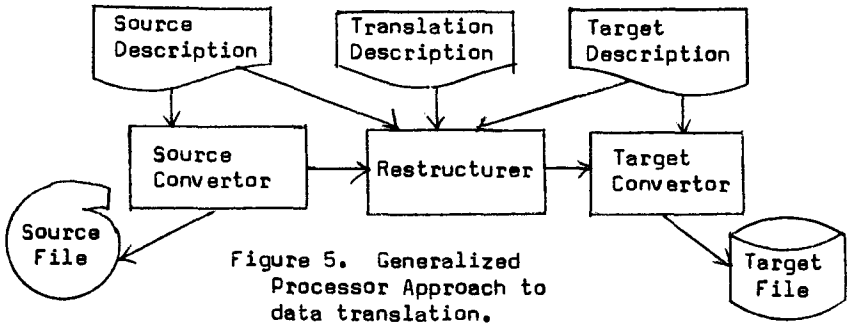

Figure 5. Generalized Processor Approach to data translation.

turer and convertors are each data translators. The source convertor translates the source from its input form to the normalized form. The restructurer translates from the input normalized form to an output normalized form. The target convertor translates from the output normalized form to the final output form. The advantages of this approach result from the use of a normalized, machine-independent form for performing the actual restructuring of the data, and the separation of the restructurer from the hardware dependent convertors. The restructurer is thus completely machine independent and can be run on any system. Only the convertors need to be tailored to particular systems. This situation makes this approach suitable for developing for use in real-time network applications. Convertors can reside at local nodes and the restructurer whereever convenient.

The compiler-compiler approach is the strategy used at the University of Pennsylvania. It is oriented towards producing efficient translation. The processor accepts as input the source, target and translation descriptions, and outputs a special purpose translator for translating all sets of data satisfying the source description. In its current version, it produces translators coded in PL/I. Later versions will produce translators coded in other programming languages. The advantages of this approach are the smaller size of the resulting translator and the efficiency of a special purpose processor generated in an automatic manner. This approach promises to figure in research on the automatic generation of information systems. This will be discussed in more detail later in this paper.

The suitability of the Generalized Processor approach for one-shot applications results from the fact that the translator can perform the translation directly, whereas, the processor of the compiler-compiler approach must first generate a specialized translator which will then only be used once. However, in the multiple-shot applications, once the compiler-compiler processor has generated the specialized translator, it can perform translations more quickly and using less space than the Generalized Processor approach.

Information systems from a data description point of view

From the applications we have seen, it is not surprising that a data translator could be used to extend the capabilities of an Information Storage and Retrieval system, in terms of data preparation and restructuring. What is perhaps surprising is that an Information Storage and Retrieval system itself is nothing but a collection of specialized data translators and data description languages. We will show how this fact can be exploited to extend the capabilities of an Information Storage and Retrieval system by rethinking its internal structure rather than by adding further external processors.

In this light, we examine query languages and the create, retrieve and update functions of an information system.

A query language is the combination of a specialized data description language and a translation description language. It is used to describe both the target structure of the data to be retrieved and also criteria for determining which data are to be retrieved. Consider a typical query: "Retrieve all authors who wrote books about information retrieval in 1973, and also the names of their publishers". The phrase " ... all authors ... and also the names of their publishers" indicates that records consisting of single "author" fields followed by possibly repeating "publisher" fields are to be output. This is a data description. The phrase " ... books about information retrieval in 1973 ... " indicates that the source of those "author" fields are records in which an index word field has "information retrieval" as a value and the date field has "1973" as a value. This is a translation description.

The create function is a data translator that translates raw data into the structure of the data base.

The retrieve function can be seen as a degenerate data translator in which the target file it constructs is always a subset of the source file. The incoming query is decomposed into its data description and translation description components, and these in turn are passed to the DSS and the TC respectively. The main role in the translation is of course played by the DSA, which must use the structure of the source file to locate the records for the output subset.

Updating is composed of the retrieve function together with a record level data translator. In this case the source files consist of the existing file and a file of update information, while the target file is constructed by modifying the existing file. First, a source data record is retrieved as discussed above. Then this record is translated by placing updated values in appropriate fields, before being returned to the original file.

Recognition of these relationships between data translation and information system functions suggests several ways in which ddl and data translation technology may influence information system design and implementation. One of the most interesting benefits may arise from "automatic programming".

We will show how, by using both a compiler-compiler type of data translator generator and a generalized data translator, the the main modules of an information system can be programmed automatically. The data translator generator can produce the "create" module automatically if it is given descriptions of the raw data format and the intended data base organization. The "retrieve" module can be implemented by combining two components. First, the translator generator is used to produce a DSA specialized to operate over the intended data base organization. Then a generalized data translator is used to supply the TC and DSS functions. These two components are then combined to produce a hybrid data translator. The TC and DSS processors can operate from the translation and data descriptions (which are generated by a retrieve or update request) to drive the specialized DSA to retrieve the required records. The record level translator required for the update module may be produced as a specialized translator using the data translator generator. In this case the data translator generator needs descriptions of the data base and the update file.

Data translation technology also suggests a solution to a frequent problem in both the Information Storage and Retrieval and the DBMS area. This problem is one of "local" inefficiencies that are felt by data base users who have occasion to concentrate their interest on various subsets of the data base. These users may require access patterns that are not optimized in the context of the current data organization. A solution to this problem (for cases where the data base is not highly linked and users partition the base) is to provide as many different "local" access structures as are needed at a given time [Wilkes]. A generalized data translator and a data translator generator can be used in a manner indicated below to create local structures and the processors needed to support them.

As usage statistics (or a given user) indicate that access performance for a particular subset of the database is unsatisfactory, a new data structure can be developed to optimize data accesses to this subset. A description of this new structure can be input to a generalized data translator which will convert the data in the subset to the new structure. A description of this new structure can then be input to a data translator generator which will produce a special purpose translator that is a "local access method" particular to the subset of the database in question. This local access method can be used directly by the user who "requested" the improved structure. Data in the subset could still be made available to "uniniated" users (i.e., users unaware of the change) via the generalized data translator, if the new and old descriptions of the subset are maintained. However, data access becomes less efficient for these uniniated users. This approach is only warrented if such accesses are infrequent and users can tolerate the inefficiency. In some cases improved efficiency could be obtained by using a hybrid data translator that has a DSA which is specialized to the new structure of the subset (i.e., which is based on the new "local access method").

Conclusion

We have reviewed some of the ideas and techniques behind generalized data description languages and automatic data translation. As with all processors that deal with descriptions of objects rather than with the actual objects themselves, there is a tradeoff between efficiency of the processor on one hand, and ease of production, convenience and flexibility on the other hand. We have indicated several problems in the information storage and retrieval area that

could benefit from increased flexibility and the rapid production of specialized processors. We therefore, anticipate some interesting developments in information systems based on the exploitation of the growing data description technology.

Bibliography

Altman — Altman, E. B., et al.: "Specifications in a Data Independent Accessing Model," Proceedings of the 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control. Denver, 1972, pp. 363-381.

CODASYL — Data Base Task Group: CODASYL Data Base Task Group Report. New York, ACM, 1971.

SDDTTG — Fry, J. P, et al.: An approach to stored data definition and translation," Proceedings of the 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control. Denver, 1972, pp. 13 - 56.

Fry — Fry, J. P., et al.: "A developmental model for data translation," Proceedings of the 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control. Denver, 1972, pp. 77 - 106.

Olle — Olle, T. W.: "A taxonomy of data definition languages," in File Description and Translation, a publication of ACM-SIGFIDET, August 1969, pp. 123 - 130.

Smith — Smith, D.: An Approach to Data Description and Conversion, Ph.D. Dissertation, University of Pennsylvania, 1971.

Smith 1972 — Smith, D.: "A method for data translation using the Stored Data Definition and Translation Task Group languages," Proceedings of the 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control. Denver, 1972, pp. 107 - 124.

Standish — Standish, T. A.: A Data Definition Facility for Programming Languages, Ph.D. Dissertation, Carnegie Institute of Technology, 1967.

Taylor — Taylor, R.W.: Generalized Data Base Management System Data Structures and their Mapping to Physical Storage, Ph.D. Dissertation, University of Michigan, 1971.

Wilkes — Wilkes, M.: Special Lecture at the University of Utah, 1972.

QUESTIONS

Willard Draisin:

Could you show us some examples of the language that was used in the implementation?

Smith:

I do not have slides made up for it, but I can show some examples afterwards. There are statements existing at each level. The conceptual level, which I mentioned decomposes further to describe the structure of items, groups, records, etc. There is a level for describing associations across records and then a series of characteristics which describe how such a structure would actually be implemented as a bit string, that is, saying how a particular type of association would be implemented by pointers, descriptors for describing how those pointers are implemented, descriptors for saying how a particular type of relationship could be implemented in terms of index tables, and where the index tables could be treated as a separate type of data structure.

Mitchell Krasny:

I am hung up on the last example which you used, where you inverted book files to abtain author files. Is it the intention of this language to do that in one pass, or is that a two-step process?

Smith:

It depends on the actual implementation, i.e., it depends on how a processor is implemented to translate those statements. The language itself does not imply any particular type of processing method or algorithm.

Mitchell Krasny:

It seems as though it would be extremely difficult to do that with a fairly large file in one pass.

Smith:

Certainly.

Mitchell Krasny:

Is your system going to have the capability to accept two input files to produce one output file?

Smith:

Yes, you could treat that as a group of separate descriptions for each file or treat the whole thing as one file with separate parts. The treatment is really quite independent of the language, and the language has to include a set of reference statements which allow you to refer to different records that could easily be in different files.

Patrick Mitchell:

In the way of a comment rather than a question, it would be relatively easy to implement as part of this system a generalized data validation subsystem.

Smith:

Yes, and as it turns out the implementation at the University of Pennsylvania did just such a thing. In effect the validation problem could be considered part of the translation/description problem in that one of the conditions that you described for selecting data is that it meets certain criteria.

Richard Guertin:

Are you familiar with the work that's being done at Stanford already on this in SPIRES?

Smith:

No, I was hoping to have a chance to talk with you about it.

Guertin:

I think that you would find that it is exactly the same thing and is already operational.

Smith:

Well, as I mentioned we already have two prototypes running, one at Michigan and one at the University of Pennsylvania.

DISCUSSION SESSION

Smith:

During the coffee break, someone mentioned their unfamiliarity with the group in CODASYL that I mentioned in my discussion. Most of you are familiar with the DDLC group that wrote the data base task management report, and are not familiar with the group that I have mentioned which is the DDL group. The two groups were formerly "cousins" operating under the same systems committee, but now the DDLC is a committee rather than a group. With respect to the relationship between the two groups, they are concerned with creating a data description language for end users, and within that language one can describe conceptual structures. Implementation structures have been reduced in their specifications, but some options are still available. My data description language concerns, and those of the group of which I am a member, go down beneath that, and we are really not concerned with how the user sees the data. Rather, we are concerned with how the data are represented on a device. Consequently, our efforts are aimed primarily at system designers and implementers.

Ben Mittman:

Can you give us some idea of what the language looks like, as it was implemented at Pennsylvania, or can you give us some description of what is planned?

Smith:

I described, I believe, five different levels of description so that perhaps I can work through an example at each level. Let's take something simple like a COBOL record. The actual description of the items and their relationships would be described almost identically as they are in the COBOL data division. Then specifying the implementation of that record as a bit string would require a series of characteristics specified for each field and each group within the record to describe how that particular piece of data was linearized. For example, the value "Jones" appearing in a NAME field would be described by; (1) type of field (either fixed or variable), (2) if the field were variable, then a delimiter string, and (3) the encodement of the value itself, e.g., bit string, ASCII, etc.

Willard Draisin:

Suppose the field length is specified as another field?

Smith:

Then there would have to be a count specified in a field above that one, which is the way it is handled in COBOL. You essentially have a repeated field. The encodement of the count field would then have to be described. Now the fact that the count field contained the length of the data field is not information that would be needed until you reached the parsing of the record. So that when you describe the length, which would be one of the characteristics of the NAME field, you would have to describe that it was a function of the field called COUNT. This kind of context-sensitive capability must be present so as to allow the specification of characteristics as being derived or variable. All four of these data description languages do contain the capability to allow you to define a characteristic in terms of any data field value or any other of the characteristics. So this is the kind of specification that results in describing the linearization of the logical structure into a bit string, and you have an analgous process in describing the kind of storage structure on whatever device is chosen. For example, the selection of tape for the storage of a COBOL file would require some specification of blocking for that tape, count fields if variable records sizes are used, and the presence of headers or trailers, and these must be described in the same sort of constructs that you used for the record description. Note that the essential relationship is an association among items, and we have been careful to keep the level identified. All of

us chose to implement the processor and the language at these separate levels. It is possible to do all of the linearization, parsing, and generation machine independently, but when you get to a specific device then that is so hardware dependent that you cannot generalize it to the same extent as the rest of the processor.

Willard Draisin:

Are you able to specify a field that is in several data records on input and exists as a single data item on output?

Smith:

Yes, that is the translation process. That's what the translation description language does.

Willard Draisin:

For example, suppose you have a field that extends over two or three data cards?

Smith:

There you have a case of a storage mapping, then a data structure. That is still one field; it simply extends over two storage cells in effect. We would be describing that at neither the logical access level nor the physical access level, but we would be describing that where we distribute the bit string over the slots. We would have to describe in the language whether the distribution would be across blocks or wholly within blocks. We would have to have some kind of SPANNING statement, and these languages do have SPANNING statements.

Ben Mittman:

How complicated an information structure can you handle? For example, can you go to networks?

Smith:

The languages do; the implementation prototypes do not. We can go to tree structures within the record, but that is it. The actual physical processors, the software machines, only handle sequential files both at Michigan and at Pennsylvania. And the extensions that are anticipated are not toward more complicated file structures but toward expanding the types of devices that can be described. Currently, it is believed that applications of data translation are not toward restructuring the data so much as toward placing the data on a different type of device.

Alan Beals:

Are all the different levels of specification required? For example, what I have in mind is the mapping of a simple COBOL created disk file into another disk file on the same machine to be accessed by COBOL programs only. Do you still have to go through all these levels of specification?

Smith:

No. Don't forget that the storage structures are exactly the same. You have used the same data management system if you're on the same machine most likely, and once you have described the storage structure demanded by that data management system component of the operating system, then you have described it independently of the data. The fact that you are changing the structure of the records within that storage structure does not change the storage structure at all. The same bottom half of the description would then be used.

Alan Beals:

I'm not sure I understand; I don't want to have to respecify the bottom half.

Smith:

If you're going to use this kind of system, it would have to be specified at least once. In the case you described, you might wish to simply use COBOL as a part of the data management system for that machine and perhaps you wouldn't have to go through the language translator.

Richard S. Marcus:

Do you worry about transformations on the data itself? For example, if you had a journal title in coded form and you wanted to translate it into its full specification?

Smith:

Yes, we can handle that simple kind of translation where it's simply a matter of encoding. What we have not incorporated is the ability to define functions over the values of the data. A simple encodement that can be handled by a table is possible in all the languages. In fact, you could define any type of encodement translation that you wished, provided it could be specified in a table.

Mitchell Krasny:

My understanding is that I could take a file that is written by a UNIVAC 1108 on a UNISERVO 3A tape in field data and, on a computer that could handle that tape drive and an IBM type tape drive, convert the field data to ASCII, 6-bit to 8-bit code.

Smith:

That is the intended application of this type of data translation. There are still in the prototypes severe restrictions on the range of device types that can be used. Most are currently tape oriented although the one at Michigan is extending to handle simple disk structures.

Mitchell Krasny:

This is intended to relieve me of the tedium involved in writing a program to do this kind of thing?

Smith:

Yes, this is in effect what people have to do now whenever they change from one system to another.

Mitchell Krasny:

Our problem is a little broader than that since we at NTIS receive tapes from many different sources and would like to make them available in some common form on a common medium.

Smith:

Another natural application of this is data translation networks, where you have different types of architectures. In this case you do not wish to one-time translate, but you wish to be able to do it dynamically. Efficiency is of course a problem here.