DADES/GP: A PROTOTYPE GENERATOR FROM DEDUCTIVE CONCEPTUAL MODELS OF INFORMATION SYSTEMS.

Jaume SISTAC
Antoni OLIVE

Facultat d'Informàtica. Universitat Politècnica de Catalunya
Pau Gargallo, 5. 08028 BARCELONA

ABSTRACT
We show that deductive conceptual models of information systems are a valid alternative to the usual operational models, and that they offer a number of advantages. We describe the main features of DADES, as a particular example of deductive language. We also present the architecture of DADES/GP, an automatic prototype generator from models written in DADES.

1. INTRODUCTION

In this paper we present the main results of our work towards the definition of a deductive conceptual modeling language (DADES) and the development of a prototype generator from specifications written in that language (DADES/GP).

The main motivation of our work has been to explore a new approach to conceptual modeling of information systems that can overcome the weaknesses of current modeling approaches. On the other hand, we have tried to analyze the feasibility and practical application of prototype generators with this approach in software development environments.

Current approaches can be characterized in terms of the framework shown in Figure 1, which is somehow based on the ISO proposal. In this framework a distinction is made between the Universe of Discourse (UoD) and the Information Base (IB). The UoD at time t consists of all those real world entities which exist at t. Knowledge about these entities is in the IB at time t, or IB(t).

IB(t) can be decomposed into two disjoint subsets: the Basis B(t), and the Derived subset, D(t), such that B(t) U D(t) = IB(t). B(t) contains all elementary informations of IB(t), while D(t) comprises all informations which can be derived from B(t) and/or D(t), in the way defined by the corresponding derivation rules.

The UoD changes over time due to external events. Occurrence of an event at time t induces a transition IB(t-1) --- IB(t) to reflect the change in the UoD.

Figure 1. Framework of conceptual models.

The conceptual model of the information system consists then of three components:

1. Information base schema
   a) Information types and structure
   b) Static and dynamic constraints
   c) Derivation rules

2. Transitions schema
   a) Event types
   b) Event constraints
   c) Effect of the events on the information base

3. Output requirements

During the past years, many languages for conceptual modeling have been developed. These languages differ in many respects, but a vast majority of them share the same approach to the modeling of the transitions schema. We call it the "operational" approach, because it consists in associating an operation to each event. Event constraints are defined as pre-conditions for the operation to be effective, in such a way that if pre-conditions are satisfied execution of the operation will preserve the static and dynamic constraints of the information base. The effects of the operation are defined by means of altering primitive operations such as insert, drop and modify, which change the contents of the IB. It is assumed, (the "frame" rule /2/) that unchanged informations will remain in the new IB state.
In /f/, we have analyzed the operational approach from several points of view, and we have found that the main weakness of the approach is that it forces the designer to make a decision on the contents of the basis, which is assumed to correspond to the information to be explicitly stored in the database. This, however, is a technical decision, which is made taking into account several implementation-dependent factors that are not part of the concept design phase. On the other hand, this decision is very sensitive with respect to some changes. Including a finer, more detailed view of the information into the base schema might involve a "drastic" change in the whole conceptual structure, it lacks extensibility (in the sense of /f/).

Examples of languages with the operational approach are SQL-5/, SQL2 /f/ and PAR1B /f/, but it is so extensive that we could rather term it as the "operational paradigm".

DEDUCTION CONCEPTUAL LANGUAGES, such as CBAI5 /f/, DATES /f/ and IP2 /f/, take another approach in several aspects, but mainly in what respects the relations schema. We call it the "deductive" approach because it relates with deductive databases /f/. We will try to show that it has some advantages and that it provides a sound theoretical basis upon which new developments can be built.

2. THE DEDUCTIVE APPROACH

In this section we review some of the main features of the deductive approach. See /f/ for further details.

2.1 Information base schema

Time plays a major role in the deductive approach. Every possible information about the DoD is associated with a time point. If(t1) which states the time when the information holds in the DoD. It can be called the occurrence (or observation) time. This time point is a component of the information itself. The information is self-describing in this respect. We will assume that all concepts are always expressed in a unique time unit (second, day, year), equal or avoid ambiguities. Thus, for example, assuming as time unit a day, "the quantity on hand of product ABC at time 1987/07/12 is 1000 units" = 1987/07/12, and "total sales of product ABC during 1985 = 1985/07/31."

By line open T of an information system we define the time during which the system operates or exists. It can be seen as a three-dimensional event 0<0(T1,...,Tn) expressed in an initial and final time, respectively, and where each tE is expressed in a unique time unit. We can then say that, for any information A(T1,...,Tn),

There is a state IB(t) of the information base for each t. Each state IB(t) is assumed to contain not only the relevant information about the DoD at t, but also the information about events occurred at t. To be valid, informations of IB(t) must satisfy a set of constraints. Each constraint may involve informations held at one or more states. Thus, the distinction between static and dynamic constraint is not made.

The information IB(t) are classified into base and derived. Base informations are those that correspond to the events that have occurred at t. All other informations are derivable from base and/or derived informations, using derivation rules. We also assume that each information type can be classified as base or derived, depending on whether its informations are base or derived.

2.2. Transitions schemas

The transitions schemas is implicit in the deductive models and, thus, it does not need to be defined. There is a transition for each t. IB(t) includes the informations about the events occurred at t. If no events have occurred at t, then, by definition, IB(t) is empty. IB(t) is given by the derivation rules.

2.3. Output requirements

The system must be able to produce a set of informations. The given by an output query is produced under a set of informations (q) is a set of informations /1, 12,7/.

3. THE DATES LANGUAGE

DATES is a language designed for the specification of deductive conceptual models. The main feature of the language is the use of the relational data model for the model of the information, and the use of the relational algebra for the definition of information sets. However, it is important to notice that the language features are self-describing in this respect. We will assume that all concepts are always expressed in a unique time unit (second, day, year), equal or avoid ambiguities. Thus, for example, assuming as time unit a day, "the quantity on hand of product ABC at time 1987/07/12 is 1000 units" = 1987/07/12, and "total sales of product ABC during 1985 = 1985/07/31."

By line open T of an information system we define the time during which the system operates or exists. It can be seen as a three-dimensional event 0<0(T1,...,Tn) expressed in an initial and final time, respectively, and where each tE is expressed in a unique time unit. We can then say that, for any information A(T1,...,Tn),

There is a state IB(t) of the information base for each t. Each state IB(t) is assumed to contain not only the relevant information about the DoD at t, but also the information about events occurred at t. To be valid, informations of IB(t) must satisfy a set of constraints. Each constraint may involve informations held at one or more states. Thus, the distinction between static and dynamic constraint is not made.

The information IB(t) are classified into base and derived. Base informations are those that correspond to the events that have occurred at t. All other informations are derivable from base and/or derived informations, using derivation rules. We also assume that each information type can be classified as base or derived, depending on whether its informations are base or derived.

3.3 Relations

This part models the structure of the informations in the Information Base. As said, this is a relational model for this purpose. The Information Base contains a set of third normal form relation schemes. Each base (from events and derived relation schemes) is defined in the same way.

In the example we have:

relation PRODUCTS (PRODUCT, MINIMUM-QTY, QTY)
key PRODUCT, key QTY.
relation ORDERS (PRODUCT, ORDER, QTY, ORDER)
key PRODUCT, key QTY, key ORDER.

Below-Minimum models informations about products that have a QTY in QOH below MINIMUM-QTY in PRODUCTS, at a specific TIME. It is assumed that a given PRODUCT appears in BELOW-MINIMUM each TIME when the above condition is met, and that a given PRODUCT associated with a "month" will also be included (see 5.4 and 5.7).

3.4 Constraints

In this part we define the constraints of the informations in the Information Base. Both static and dynamic constraints are defined in the same way, thus providing a uniform construct in both cases. (This is in contrast with the operational approach where static and dynamic constraints require different formalisms.)

A constraint definition consists of three parts:

1. Domain
2. Condition
3. Existence condition.

The existence condition defines the condition that must be satisfied, while the domain states when this condition must hold.

An example could be:

constraint PRODUCTS-IN-QOH
for each product p, if exist exists PRODUCTS (PRODUCT-p).

The constraint states that for any product p present in relation QOH (domain p) the condition (PRODUCT-p) must exist (must not be empty). Here, PRODUCTS (PRODUCT) = (p | QOH PRODUCT(p)).
In Session 19, we have analyzed the operational approach from several points of view, and we have found that the main weakness of this approach is that it forces the designer to make a decision on the contents of the basis, which is assumed to correspond to information that is to be explicitly stored in the database. However, this, in turn, generates a design that must be made independently of the conceptual phase. On the other hand, this decision is very sensitive to changes, including an independent, more detailed type of information into the information base schema. This is the single feature that is most difficult to change in the whole conceptual model. Thus, the probe extends the content of the operational phase.

Examples of languages with the operational approach are SHRM/5, GDBS/6, and PABLS/7, but it is also important to note that we could then have an operational phase.

Deductive conceptual languages, such as DIAM/8, DANOIS/9, and IML/10, can be considered as the operational phase. They take the conceptual model and extend it, especially in what respects the translation scheme. We call it the "deductive approach" because of its similarity with deductive databases /10/. We will try to show that these advantages and that it provides a sound theoretical base upon which new developments can be built.

2. THE DEDUCTIVE APPROACH

In this section we review some of the main features of the deductive approach. See /5/ for further details.

2.1 Information base schema

The time plays a major role in the deductive model. Every time a possible event is that the time point T(t1), which occurs the time when the information event is in the database. It can be called the occurrence of an event. This point is a component of the information event, which we see as self-describing in this respect. We will assume that the times are always expressed in a unique time unit, seconds, for example. It is convenient to avoid ambiguities. Thus, for example, assuming an area unit a degree, the quantities of area and volume of area are expressed in a unique time unit, seconds.

3. THE DASDBS LANGUAGE

DASDBS is a language designed for the specification of deductive conceptual models. The main features of the language are the use of the model with the information base, and the use of the associated relational database for the definition of information sets. However, it is also important to have certain features for dealing with practical aspects, such as the time handling and input/output specification.

A DASDBS conceptual model consists of seven parts:

1. Time units
2. Domains
3. Relations
4. Constraints
5. Derivation rules
6. Input/output operations
7. Outputs

Where the items (1) - (5) correspond to the information base schema.

Below, we describe some of the aspects of the language model example used to illustrate the features of the language.

2.2 Time units

In this part we define the system time units and the aggregated time units that are required in the model. In the example we could define:

Time units SECOND, MONTH.

Time units SECOND, MONTH mean that the smallest time unit is "second" and the aggregated time unit "month" will also be used (see 3.4 and 3.5).

3.2 Domains

This part defines the domains of the attributes of the relations in the information base.

In the example, we have:

domain ORDER is INTEGER.

ORDER is between 1 and 1000.

domain PRODUCT is CHAR(5).

Domain corresponding to the time units are implicit and need not be defined. It is assumed that each time unit is represented in its complete form, such as year/month/day/hour/minute/second for second.

3.3 Relations

This part models the structure of the information base for the information base. As said before, we use the relational model for this purpose. The Information Base consists of a set of time units and their corresponding relation schemas. Both base (from events) and derived relation schemas are defined in the same way.

In the example we have:

relation PRODUCTS (PRODUCT, KEY PRODUCT, QIH, PRODUCT, TIME, QIH, KEY PRODUCT, TIME, KEY PRODUCT, TIME).

relation REFILLMENTS (PRODUCT, TIME, KEY PRODUCT, TIME, KEY PRODUCT, TIME).

relation ORDER (ORDER, TIME, PRODUCT, KEY PRODUCT, TIME).

3.4 Constraints

In this part we define the constraints of the information base. Both static and dynamic constraints are defined in the DASDBS, thus providing a uniform construct in both cases. This is in contrast with the operational approach where static and dynamic constraints require different formalisms.

A constraint definition consists of three parts:

1. Name
2. Domain
3. Existence condition

The existence condition defines the condition that must be satisfied, while the domain conditions when this condition must hold.

An example could be:

constraint PRODUCTS-QUH PRODUCT = QUH PRODUCT

The constraint states that for any PRODUCT and QUH PRODUCT, if there exists a PRODUCT, then the existence condition holds.

The constraint states that for any PRODUCT and QUH PRODUCT, if there exists a PRODUCT, then the existence condition holds.
Other examples are:

\begin{verbatim}
constraint NO-ORDERS-WHEN-BELOW-MINIMUM
for all products p, time t in orders
not exists below-minimum(p,t)
\end{verbatim}

where (p,t) is an abbreviation for (PRODUCT, time-t).

\begin{verbatim}
constraint AT-LEAST-4-REPLENISHMENTS-PER-MONTH
for all months m in months
for all products p in products
exists q in replenishments(p,m,tm,mn)
which means that for any cp,p,m months x products[product] must exist at least one replenishment of product p during month m. Note the use of the natural join operator (and) on common attributes (time in this case).
\end{verbatim}

3.5 Derivation Rules

Relation schemas defined in the Information Base are classified into base and derived. For each derived relation schema one or more derivation rules must be defined. They state how information are derived.

A derivation rule consists of five main parts:

1. Name
2. Domain
3. Function
4. Contents
5. Precedents

The domain defines the extent over which the rule is applied, the function indicates the derived information and the function gives the name of a function that applied to the precedents produces the derived information.

In the example, QOH and BELOW-MINIMUM are derived relations. The derivation rule for the latter could be:

\begin{verbatim}
constraint BELOW-MINIMUM
for all products p in products
for all months m in months
for all periods p in period
for all time t in times
exists below-minimum(p,t)
where quantity in QOH is less than minimum quantity in products p
\end{verbatim}

The meaning is as follows. The rule is defined (domain) for every pair:

\begin{verbatim}
c,p,m TIMES x PRODUCTS[PRODUCT]
\end{verbatim}

Variables t and p act as parameters. The information derived is given by the relational algebra expression

\begin{verbatim}
BELOW-MINIMUM(p,t), which refers to a tuple of products(p) in the product database. The precedents are the parameters themselves, the quantities in QOH, and the product information of p. t is the name of the function, which is defined informally in the "function in" part.
\end{verbatim}

Input contents is specified in the "gives" part. Thus, in the REPLENISHMENT-RECEIVED input type it is defined on the occurrence of this type for each event, with initial data attached to PRODUC and_TIME, and that the information is transmitted to the following REPLENISHMENT(p,t), where p and t are parameters. Note that the parameters allow us to relate the domain with the contents.

In the INITIALIZATION case, each occurrence (there may be only one) gives the contents of PRODUC and_TIME.

3.7 Outputs

This last part of the model specifies the outputs the system has to provide. For each output type we define:

1. Name
2. Domain
3. Contents

where the domain describes the condition that produces an output occurrence, and the contents defines what information is given each occurrence.

Usually, outputs must be produced on request from the users. In this case, the domain specifies that each request or output is an output occurrence. For example,

\begin{verbatim}
output QTY-ON-HAND
for all products p
for all time t in times
for all orders o in orders
for all products p
\end{verbatim}

where for each request of type QTY-ON-HAND (identifying attribute is TIME) the system must give the quantity on hand and product information, at the time of the request, for all products.

Very often, however, outputs must be produced when some internal condition holds. The domain can also be defined in the domain. For example,

\begin{verbatim}
output PRODUCT-BELOW-MINIMUM
for all products p
for all time t in times
for all periods p
for all periods p
\end{verbatim}

where which gives that whenever a product is below minimum, the system must provide the informations given by products(p).

Conditions associated with time can also be defined. For example, a monthly report could be defined in the following way:

\begin{verbatim}
output MONTHLY-REPORT
for all products p
for all months m
for all periods p
\end{verbatim}

Note that an output (input) is defined in base to the information schema, independently of output (input), and that it is not necessary to relate outputs with the generation rules required to produce them. This allows the possibility that users and/or different people work in parallel.

4. THE DADSS/GP PROTOTYPE GENERATOR

Several attempts to generate prototypes from operational models have been made, such as the one described in /12/ for GIST models. To our knowledge, the only similar work done for deductive models is the PROLOG implementation of IFL models /15/. It is important to use PROLOG as the implementation language of deductive models, because of the ease of translation of derivational rules, constraints, etc. into PROLOG constructs. However, as is pointed out in /15/, PROLOG lacks an adequate treatment of dynamics, which is essential in our context. On the other hand, such an implementation does not allow us transferring a conceptual model into an efficient architectural model.

We regard prototype generation both as a practical application and a theoretical challenge. By the forner we mean that prototypes have a practical value as the only means to verify the correctness of the conceptual model with respect to the user's perception of the reality. By the latter we mean that efficient prototype generation poses all the problems the designer has to face when he tries to implement a conceptual model.

Figure 2 shows the global architecture of DADSS/GP, which has been implemented on a VAX/VMS with the 9th relational database system. The system consists of three major components: Analyzer, Scheduler and Generator. The Analyzer interacts with the user, receives, verifies and stores the DADSS conceptual model. Once the definition is complete and correct the prototype can be generated. We do this in two phases. In the first, the Analyzer defines the database schema and determines the sequence in which the actions (inputs, derivations, rules, outputs) have to be performed. Several sequences are usually possible, and optimization procedures have their place here. The Analyzer also simulates the generator, which is then compiled and put into execution.

PASCAL has been used as the implementation language for both the DADSS/GP system and the prototype. The designer also writes the functions of each algorithm in that language, using program skeletons produced by the Generator.

The prototype iterates for each time of the life span. At the end of the first phase of an iteration, the prototype asks the user for the current time of the month, etc. This allows the user to simulate in a few iterations the whole life span.
3.5 Derivation Rules

Relation schemes defined in the Information Base are classified into base and derived. For each derived relation schema one or more derivation rules must be defined. They state how information is derived.

A derivation rule consists of five main parts:
1. Name
2. Domain
3. Intension
4. Function
5. Precedence

The domain defines the extension over which the rule is valid. The intension indicates the derived information and the function gives the name of a function that applied to the precedents produces the derived information.

In the example, QEH and BELOW-MINIMUM are derived relations. The derivation rule for the latter could be:

```
constraint BELOW-MINIMUM for all PRODUCT p in PRODUCTS

- p.QOEHP <= p.PREVIOUS

function is / QOEHP <= MINIMUM(p)

if quantity in QOEHP is less than minimum quantity in PRODUCTS

The meaning is as follows. The rule is defined (domain) for every pair:
```

```
3.6 Inputs

This part specifies the input types of the information system. The definition of an input type consists of four main parts:

1. Name
2. Domain
3. Frequency
4. Contents

where the domain defines where the input is received, the frequency gives the number of inputs of this type per time unit and the contents describes the kind of information given by an input.

In the example there are three input types. Their definition is:
```
```
```
Input RECEIPTS-RECEIVED
- for all PRODUCT p in PRODUCTS
- p.BELONGS-MIN(p)

function is / RECEIPTS-MIN(p)

where for any quantity in RECEIPTS-MIN is less than minimum quantity in PRODUCTS

The meaning is as follows. The rule is defined (domain) for every pair:
```

```
Note that an output (input) is defined in base to the information scheme, independently of other output (input), and that it is not necessary to relate outputs to relations or derivation rules required to produce them. This allows the program to work in parallel.

4. THE DADS/QP PROTOTYPE GENERATOR

Several attempts to generate prototypes from operational models have been made, such as the one described in [12] for GIST models. To our knowledge, the only similar work done for deductive models is the PROLOG implementation of IFP models [15]. It is tempting to use PROLOG as the implementation language of deductive models, because of the easy translation of derivation rules, constraints, etc., into PROLOG constructs. However, as is pointed out in [15], PROLOG lacks an adequate treatment of dynamics, which is essential in our context. On the other hand, an implementation does not allow us transforming a conceptual model into an efficient architectural model.

We regard prototype generation both as a practical application and a theoretical challenge. By the former we mean that prototypes have a practical value as the only means to verify the correctness of the conceptual model with respect to the user's perception of the reality. By the latter we mean that efficient prototype generation poses all the problems the designer has to solve when he tries to implement a conceptual model.

Figure 2 shows the global architecture of DADS/QP, which has been implemented on a VAX/VMS with the 4th relational database system. The system consists of three components: Analyzer, Scheduler and Generator. The Analyzer interacts with database and verifies and stores the DADS conceptual model. Once the definition is complete and correct the prototype can be generated. We do this in two phases. In the first the Analyzer defines the database schema and determines the sequence in which the actions (inputs, derivation rules, outputs) have to be performed. Several actions, often usually possible, and optimization procedures have their place here. The Generator writes the prototype programs, which is then compiled and put into execution.

PASCAL has been used as the implementation language for both the DADS/QP system and the prototype. The designer also writes the functions of each action in that language, embedding program skeletons produced by the Generator.

The prototype iterates for each time of the life span. Each time, at the beginning of an iteration, the prototype asks for the current time and month, etc. This allows the user to simulate in a few iterations the whole life span.
Thereafter, the order of the actions performed by the prototype is given by the Scheduler. A usual pattern is:

a) Ask the user for the events that happen at the current time and the associated input information, which is stored in the database. Constraints are verified and input rejected if fail.

b) Determine which informations are derived at the current time and call the adequate procedures to do it. Derived information is stored in the database.

c) Determine which outputs are to be produced at the current time. For each of them the corresponding information is retrieved from the database and written into an output device.

In the first version, we were only interested in proving the feasibility of the conceptual model. So, we did a naive implementation in which:

1) A "forward inference" approach was used for derivation rules. The derived information was continually maintained.

2) The database grew indefinitely because only insertions were done.

3) No attempts to optimization were done.

All of these points are concerned with the efficiency of the prototype, which might be irrelevant in some applications.

In the second version we aimed at improving the efficiency. We developed a number of heuristics which were introduced into the Scheduler. These are:

1) Combine inputs, derivation rules and outputs when possible. For example, two derivation rules with the same domain and precedents can be combined into a single derivation rule.

2) Delete informations from the database when they will not be used in other iterations. These can be determined formally. Although these heuristics are able to produce satisfactory prototypes we plan to formalize the problem (sequence optimization, database contents, derivation rules implementation) in a general framework and develop complete optimization algorithms.

5. CONCLUSIONS

We have shown that deductive conceptual modeling of information systems is a valid alternative to the usual operational approach, and that it offers several advantages, such as:

1) It does not force the designer to make an early decision about what to store in the future database;
2) It allows an easy extensibility of the conceptual model; and
3) It provides a sound basis upon which efficient implementations can be developed.

We have also described the main features of the DADDS language, as a particular example of deductive languages. Practical application of the language to real projects indicates that it may serve as a specification language, even if no prototypes are generated.

However, automatic prototype generation has a great value in practical application, as we have observed in our experience using DADDS/GP. We believe that future software development environments will incorporate some kind of prototype generators. On the other hand, prototype generation has a great theoretical interest.

ACKNOWLEDGMENT

This work was supported by the CAIDIT.

REFERENCES


3/ Olivé, A. "A comparison of the operational and deductive approaches to conceptual information systems modeling". To be presented at the IFIP Congress 86, Dublin, Ireland, September 1986.


RESUME

Nous montrons que les modèles conceptuels déductifs de systèmes d'information sont une alternative valable aux modèles opérationnels habituels et qu'ils offrent un certain nombre d'avantages. Nous décrivons les principes caractéristiques de DADDS, comme un exemple particulier des langages déductifs. Nous présentons aussi l'architecture de DADDS/GP, un générateur automatique de prototypes à partir de modèles écrits en DADDS.
Deletion of Information from the Database

Thereafter, the order of the actions performed by the prototype is given by the Scheduler. A usual pattern is:

a) Ask the user for the event that happens at the current time and the associated input information, which is stored in the database. Constraints are verified and input rejected if fail.

b) Determine which information can be derived at the current time and call the adequate procedures to do it. Derived information is stored in the database.

c) Determine which outputs are to be produced at the current time. For each of them the corresponding information is retrieved from the database and written into an output device.

In the first version, we were only interested in proving the feasibility of the prototype. So, we did a naive implementation in which:

1) A "forward inference" approach was used for derivation rules. The derived information was continually maintained.

2) The database grew indefinitely because only insertions were done.

3) No attempts to optimization were done.

All of these points are concerned with the efficiency of the prototype, which might be irrelevant in some applications.

In the second version we aimed at improving the efficiency. We developed a number of heuristics which were introduced into the Scheduler. These are:

1) Combine inputs, derivation rules and outputs when possible. For example, two derivation rules with the same domain and precedent can be combined into a single derivation rule.

2) Delete information from the database when they will not be used in other iterations. These can be determined formally.

Although these heuristics are able to produce satisfactory prototypes we plan to formalize the problem (sequence optimization, data base contents, derivation rules implementation) in a general framework and develop complete optimization algorithms.

REFERENCES


3/ Olivér, A. "A comparison of the operational and deductive approaches to conceptual information systems modelling". To be presented at the IFIP Congress 86, Dublin, Ireland, September 1986.


ACKNOWLEDGEMENT

This work was supported by the OAOTT.

Figure 2. DADES/GP Architecture