Welcome – An Architecture for Integrated Customer Care Systems

Narain H. Gehani
Bell Laboratories
nhg@research.bell-labs.com

William McKenna
Bell Laboratories

Inderpal Singh Mumick
AT&T Laboratories
mumick@research.att.com

Michael Rabinovich
AT&T Laboratories
misha@research.att.com

August 22, 1996

Abstract

An integrated customer care (ICC) system aims to provide a single point of contact to the customer for all issues surrounding a service or a product (an offer). For example, the customer care system could provide a single point of contact for all phases of a service starting from ordering and provisioning to billing and maintenance.

The single point of contact should be maintained even when the offer is put together using services and products provided by multiple providers. The providers could belong to different parts of the same company or could be outside companies. Note that an individual provider may not have a customer care system, and it is the responsibility of the ICC system to put together various systems from each provider to get a single point of contact for the customer.

A first step in providing a single point of contact for the customer is to allow a customer service agent (CSA) to access the various provider databases or systems from his or her workstation. The use of multiple windows with cut and paste facilities will be of great help to a CSA. The CSA acts as the integrator of the provider systems. It is undesireable for such manual integration to be done by the customer.

The Welcome customer care architecture automates the integration of the provider databases. For example, using materialized database views, an integrated view of the provider databases will be provided to the CSAs. CSAs will interact with one interface, and their actions will be translated into several transactions against the provider databases.

The customer will also be able to directly access (a restricted version of) this interface to order a service, check its status, look at billing information, etc., and connect to a CSA for further help. In addition, a customer and CSA could cooperatively browse relevant information. Direct access by a customer to the customer care system will significantly reduce the number of CSA needed.

1 Introduction

Customer care systems aim to provide a single point of contact to customers for an "offer", that is, a service or a product offering composed from services and products provided by multiple providers belonging to different units of the same company or other companies. Without a customer care system, a customer will have to individually contact the different providers making up the offer, perhaps even different units within a provider's organization for ordering, provisioning, billing, technical questions, etc. Moreover, the various individual providers making up the offer may separately contact the customer to get the similar or even the same information such as customer's address. Instead of dealing with multiple contact points, multiple bills and so on, a customer would prefer to deal with a single point of contact and get one bill. The customer would not like to be the one coordinating the activities of the providers making up an offer.

The customer care problem arises because of the following reasons:

1. Legacy – the offer has been hacked together. Different components of an offer are supplied by different parts of the same company; the different parts of the company operate as separate companies without an integrated view of the customer or an integrated component database.

2. Systems Integration – An offer is made up by integrating services and products from multiple companies.

In the absence of a customer care system, the customer sees the offer as coming not from a single company such as IBM or AT&T but from a group of companies. It is the customer's responsibility to find the right contact person (perhaps by contacting the "offer manager"). The customer must deal with several accounts, one each with the providers making up an offer and the bills sent by them.

It may not be feasible to integrate the providers systems into one for legacy and/or organizational reasons. Nevertheless. it is important that the customer view an offer as coming from a single provider and not from an ad hoc collection of multiple providers, each with its own customer contact point or points, billing procedure etc.

A first step in providing a single point of contact for the customer is to allow a customer service agent (CSA) to access the provider databases from the CSA workstation. The use of multiple windows with cut and paste facilities will be of great help to a CSA. Although, the various provider systems are not integrated into one system with this approach, such a customer care system represents a significant advance over the customer having to deal with multiple points of contacts - one or more for each provider. The key point to note here is that it is a CSA's function to act as the integrator of the provider systems. Thus, CSA must know multiple systems that take part in customer care.

Our focus is on the integrated customer care architecture that automates the integration of the provider databases. For example:

- Using database views, integrated views of the provider databases (IVPD) will be provided to the CSAs.

- To provision a service, CSAs will not have to execute multiple transactions against the provider databases or have to roll back some of the transactions if the service cannot be provisioned. Instead, CSAs will be able to execute one transaction against the IVPD which will be translated into transactions against the provider databases. There will be automatic undoing of the completed transactions (using compensating transactions as in "sagas") if all of the transactions cannot be completed successfully.

- Customers will be able to directly access the customer care system to order a service, check its status, look at billing information, etc., and connect to a CSA for further help. Access to the customer care system could be via a telephone (with voice recognition), via direct dialup, or via the internet.

2 Offers

An offer is a very generic term used to describe almost any product or service in the marketplace. For the purpose of this document, we try to define what an offer is in the context of a customer care system.

We will use the following terminology for defining offers and throughtout the paper:

A *company* refers to a corporate entity with a common controlling board. A *company* may be divided into several business units (BUs). Each BU consists of one or more departemnts. The word *organization* can refer to any level in the corporate hierarchy.

Subunit of an organization a refers to the corporate level below the level of a. Thus the subunit of a company would be a BU. The subunit of a BU would be a department.

Definition 2.1 (Basic Offer): A basic offer is a product or a service for which information about the customer and/or product/service is collected by the offering organization.

The product or service may be purchased, or may be given as a promotion. ■

Information about a product can include the serial number, the date of manufacturing, the date of sale, the location of sale, the characteristics of the product, the manual on the product, the sale price, the warranty, the terms of the sale, etc. Information about a service can include the terms of the service, the start date, the end date, the servicing location, the sale price, the billing mechanism, etc. Information about the customer typically includes the name, address, contact information, type of usage, etc. Besides these, information about service centers may also be stored.

A non-trivial organization would typically store all this information in databases. The information can then be used to do marketing or other studies by the organization. The information is also used to provide customer care.

Figure 1 graphs the various types of contact that a customer may have with the offering organization (or its subunits) about an offer.

Individual points in this graph can represent points of contact, or they can represent systems or processes to support these functions.

Several products and services are often put together into one integrated offer by companies. We will refer to these as offer packages. The products and services composing a package can be produced by different companies, or by different organizations within a company.

Definition 2.2 (Offer): An offer is defined recursively as either:

- A basic offer, or
- A collection of offers presented in a single package.

The offer may be purchased, or may be given as a promotion. ■

Definition 2.3 (Integrated Offer): An offer that is not basic is called an *integrated offer*. ■

With an integrated offer, information about several distinct products or services would be collected by the offering organization. The customer may need to have various types of contact for each of the products and services included in the offer, as illustrated by the following Figure 2. Each point on the X-axis in Figure 2 represents a product constituting the integrated offer. The vertical line at this point represents the several types of contact the customer needs to have for this particular product. The entire space of Figure 2 thus represents contact for all products and services composing the integrated offer.

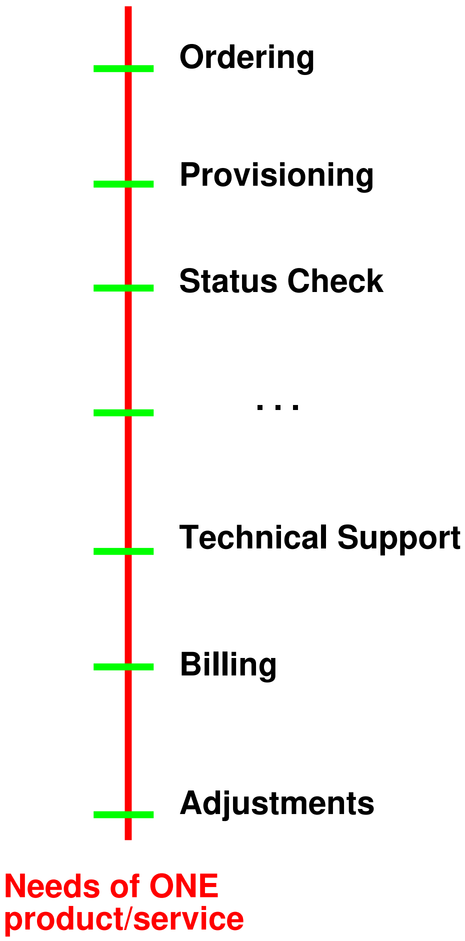

Figure 1: Customer Contact for a basic offer.

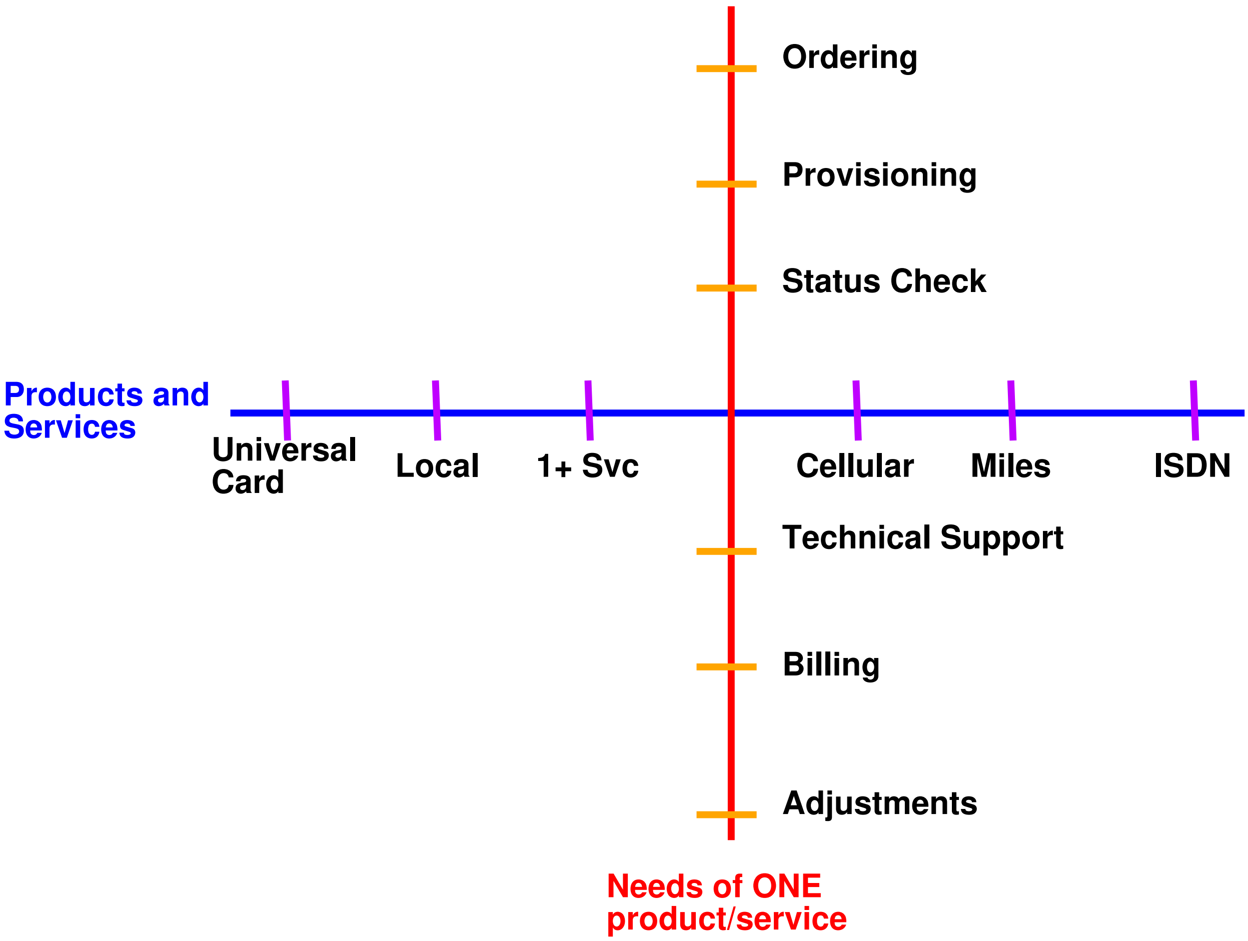

Figure 2: Customer Contact for an integrated offer.

3 The Customer Care Problem

Definition 3.1 (Customer Care Problem for an Offer): The customer care problem for an offer is to provide a single point of contact for all types of customer contacts relating to all the products and services included in the offer.

An architecture that provides a single point of contact for all types of customer contact for all the products and services included in the offer will be called an *integrated customer care architecture.*

■

3.1 Customer Care for a Basic Offer

A simpler form of the customer care problem is manifested for a basic offer, since only one product or service is involved. An architecture that provides a single point of contact for all types of customer contact for the product or service included in the basic offer will be called *unified customer care architecture*.

An organization offering a basic offer would typically divide up its work into subunits responsible for each aspect of the customer contact for the concerned product. Each of these collects and manages data for the service it provides. Solving the customer care problem for the basic offer thus requires, amongst other things, that data from several data stores be brought together into a uniform interface, and that commands issued from this uniform interface be propagated to these several data stores. It is also possible that some of the functions relating to a basic offer are contracted out to an external agency. In such a case, the external data store must also be integrated into the uniform interface.

A unified architecture attempts to fold each vertical line in Figure 1 into a single point on the Y-axis.

3.2 Customer Care for an Integrated Offer

The additional complexity of customer care for an integrated offer is that the products and services forming an integrated offer can be offered by different organizations, some or all of which may be distinct from the organization putting together the offer, and further these organizations may be in different autonomous companies. Providing a single point of contact thus means that data stores and systems from several distinct organizations would have to co-operate. Building an integrated customer care system thus poses an additional complexity that one may not have much flexibility over systems of another organization.

An integrated customer care architecture is thus a generalization of the unified architecture where the entire graph of Figure 2 has to be folded into a single point at the origin.

The goal of our work is to come up with an integrated customer care architecture that permits one to build a customer care system for a new offer, or to extend an existing system to new offers. We would also specify basic requirements for systems and data stores managing data and processes around individual points in Figure 2 so that an integrated customer care may be built.

4 Example

Consider a unit of a diversified large company (such as AT&T or MCI) that needs to present its customers (client companies) with an integrated offer that combines services and products from different business units of the large company.

The component products and services may include:

- Simple Long Distance Telephone Services.
- Programmable Long Distance Telephone Services
- Calling Card Service
- Cellular Services
- Local telephoner service.
- High End offers such as insurance, banking, ...
- Computer Hardware

For each of these components, the following items need to be coordinated:

- ordering
- provisioning
- billing (billing integration)
- maintenance
- inquiries

In the absence of an appropriate customer care system, the customer must deal separately with the various business units for the different components of the offer.

5 Generic Functionality/Features of Integrated Customer Care Systems

To provide a single point of contact, a customer care system must include the following features and capabilities:

- Customer Profiles: A customer profile must be created for each new customer. All pertinent customer information must be maintained.
- Offer recording: Every new offer must be recorded. Orders must be placed to various suppliers for equipment and services. The suppliers may or may not have electronic means by which to receive and process orders. Information must be maintained on method of installation and associated delivery schedules and locations (e.g., delivery may be to a VAR who is contracted to install a part of the solution).
- Integrated view: an integrated view on the data contained in multiple component databases must be maintained. The CSA should not have to execute transactions against the component databases. Rather, the CSA should operate on the integrated view provided by the system. Operations against this view are translated into operations against individual databases by the system.

- Work Flow: Multiple activities must be coordinated during provisioning of the offer (see Section 9). Milestones during the provisioning must be recorded. These may include delivery and installation dates, and periodic customer satisfaction reports.

- Billing support. The customer care system will receive all vendor bills. Note that customer views (profiles) may differ among vendors. Billing cycles vary among vendors, further complicating billing. A single, integrated bill must be issued to the customer. Different methods for billing have been proposed.

- Monitoring: The system must track customer interactions and service status and take appropriate actions as specified. Examples of monitoring include monthly reports to product management (orders complete on time, etc.) and incident tracking from report to resolution.

- Decision Support Tools: Across the entire customer spectrum, data should be available to list all sales leads, analyze customer satisfaction, list current customers, record sales leads, etc. This requires maintaining materialized aggregate views (see Section 8).

- System flexibility: THe system should be extendable to handle new industry types, be easily changeable to add new capabilities, etc.

6 The Welcome Architecture

6.1 Manual Integration

The first attempt at an integrated customer care system (Figure 3) essentially hides the multiple providers from the customer offering a single point of contact. No effort is made to integrate the databases of the various providers in the same company, say to provide an integrated customer profile. Also, the customer care agent has to learn several systems.

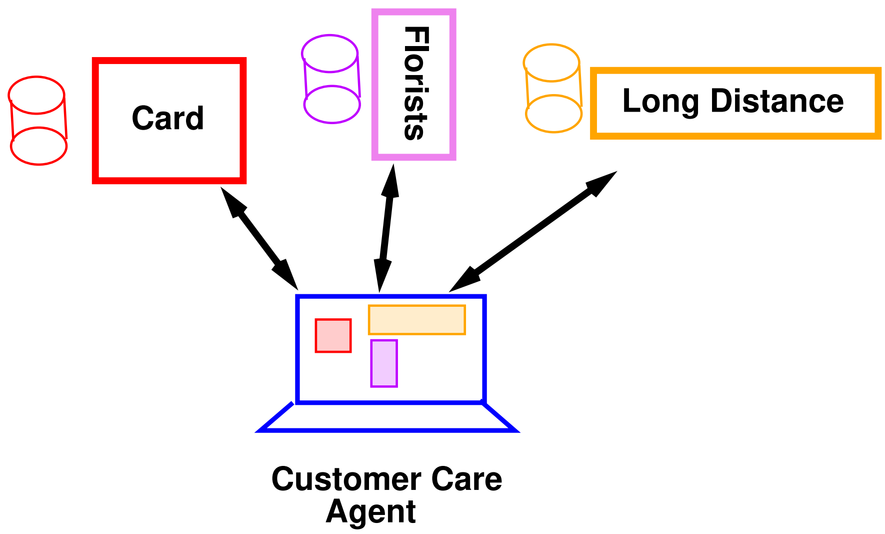

Figure 3: Customer Care Agent functions as the integrator.

6.2 Integrated Customer Care Architecture

Our goal is to focus on an integrated customer care architecture that will provide an integrated view of the offer by presenting one logical view of the offer which will also support access and updates of the provider databases. The basic architecture is shown in Figure 4.

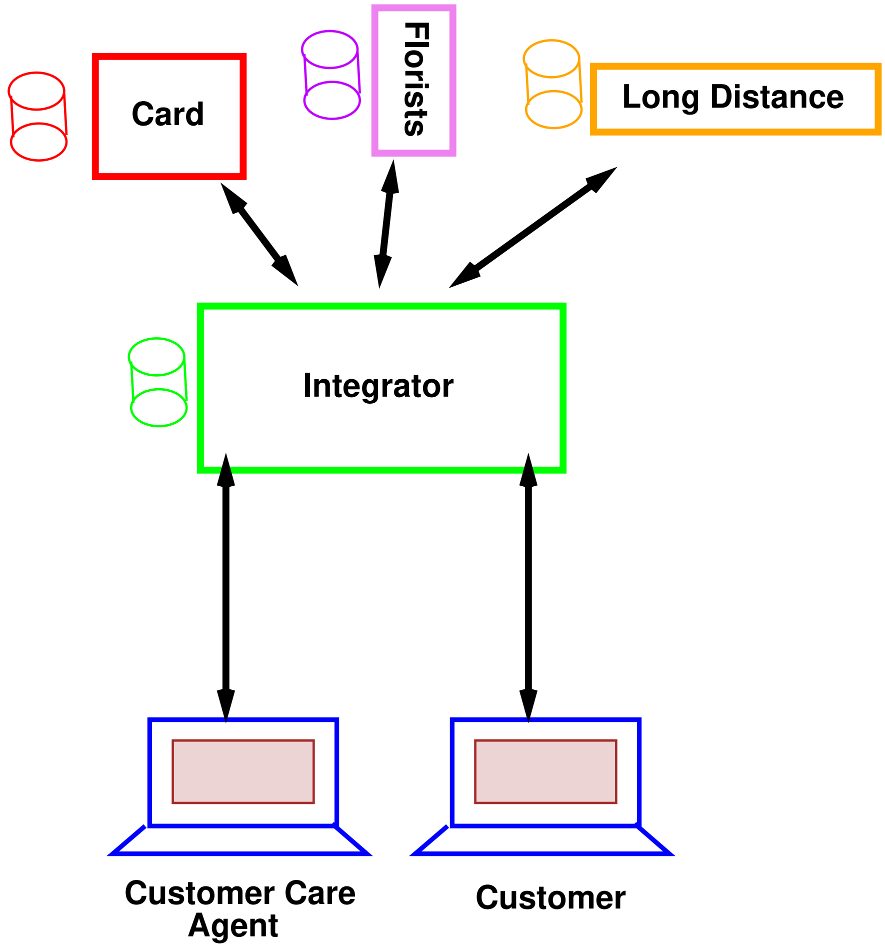

Figure 4: Integrated Customer Care Architecture.

- The component databases will be hidden from the CSAs; instead, using database views, an integrated view of the provider databases (IVPD) will be provided to the CSAs. It will appear to the CSAs that they are interacting with one database. The integrated view will be kept consistent with the provider databases using automated techniques.

- CSAs will not have to execute individual transactions against the provider databases to offer a service to a customer, make changes to the customer account, etc. Instead, CSAs will be able to execute one transaction against the IVPD which will be translated into transactions against the provider databases. Techniques, such as workflow management and "sagas", will be used to ensure consistency between the provider databases.

- Object-oriented technology will be used to build a set of objects for customer care systems. Customer care system functionality could be extended by modifying these objects. These objects could also be used to build customer care systems for different offers.
- Customers should be able to directly access the customer care system to order a service, check its status, look at billing information, etc., and connect to a CSA for further help. Access to the customer care system could be via a telephone (with voice recognition), via direct dialup, or via the internet.

7 An Integrated Customer Care System

Figure 5 shows a prototype of the Integrated Customer Care System built using the Welcome architecture.

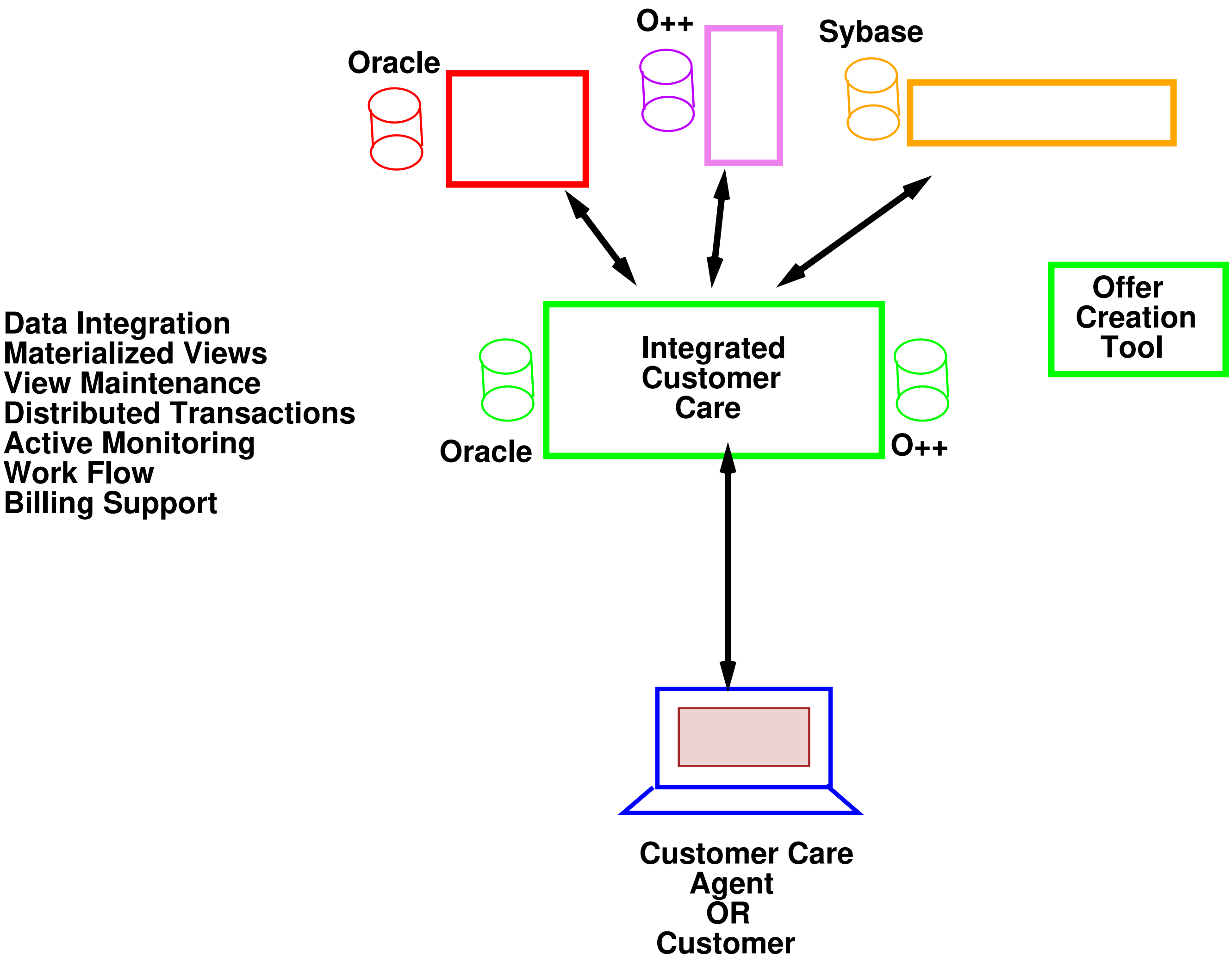

Figure 5: Integrated Customer Care System

8 Views

Views and view maintenance play a central role in the Welcome architecture for building an Integrated Customer Care system. We define some terms here:

What is a view? A view is a derived relation defined in terms of base (stored) relations. A view thus defines a function from a set of base tables to a derived table; this function is typically recomputed every time the view is referenced.

What is a materialized view? A view can be materialized by storing the tuples of the view in the database. Index structures can be built on the materialized view. Consequently, database accesses to the materialized view can be much faster than recomputing the view. A materialized view is thus like a cache – a copy of the data that can be accessed quickly.

Why use materialized views? Like a cache, a materialized view provides fast access to data; the speed difference may be critical in applications where the query rate is high and the views are complex so that it is not possible to recompute the view for every query. Materialized views are useful in applications such as customer care systems that need to provide a fast response to the customer.

What is view maintenance? Just as a cache gets *dirty* when the data from which it is copied is updated, a materialized view gets dirty whenever the underlying base relations are modified. The process of updating a materialized view in response to changes to the underlying data is called view maintenance.

What is incremental view maintenance? In most cases it is wasteful to maintain a view by recomputing it from scratch. Often it is cheaper to use the heuristic of inertia (only a part of the view changes in response to changes in the base relations) and thus compute only the changes in the view to update its materialization. We stress that the above is only a heuristic. For example, if an entire base relation is deleted, it may be cheaper to recompute a view that depends on the deleted relation (if the new view will quickly evaluate to an empty relation) than to compute the changes to the view. Algorithms that compute changes to a view in response to changes to the base relations are called *incremental view maintenance* algorithms.

What is view update? One can treat a view as if was a base relation, and specify updates, deletions, and insertions of tuples into the view. The system must then map these operations to updates, deletions and/or insertions into the base relations so that a recomputation of the view from the new base relations results has the desired affect upon the view. The problem of translating updates operations on the view to update operations on the base relations is called the view update problem.

We now describe some of the techniques to build, maintain, and update views that will be useful in building a customer care system.

8.1 Data Integration and Materialized Views

Integrating data from several providers. This is data warehousing where the warehouse is build by collecting, and integrating data from multiple systems. The integration is to be expressed as a view.

An integrated customer care system needs to bring together related data in distinct databases.

A common approach for integration is to define an integrated view, and then map queries on the integrated view onto queries on the individual systems. This model is not good for applications where response time is critical (such as a customer care system), or for decision support applications with complex queries. Further, in this model, the availability of the integrated view is the lowest denominator of the availabilities of all databases touched by a query. Finally, such query transaltion can become almost impossible when some of the underlying "databases" are actually flat files, or other non-relational systems.

An alternative model [GJM94], good for applications where fast response to queries and high availability are important materializes the integrated view and stores it in a target database. All queries on the view are then answered directly from the view. A copy of the underlying data is *not* stored in the target database. This approach solves the difficulties with the query translation approach. However, it does involve the additional storage expense for the materialized integrated view. It has been our experience that, in many applications, query performance needs alone are enough to justify this additional expense.

Changes continue to be made to the underlying database[1], and a problem that must be solved is how to maintain the integrated view current as the underlying databases change. It may be expensive or impractical to obtain much more than just a periodic report of local updates from each underlying database. Even if an underlying database does not provide relational query support, one can require it to provide a log of changes it makes.

Maintaining the integrated database current in such a world requires one to have efficient incremental view maintenance techniques. Indeed, it is important to look one step further when defining the integrated views, and define views that are maintainable in an efficient manner. Efficient maintenance means, at the very least, that the view be maintainable when a log of changes arrives from a component database, regardless of whether the component databases themselves are accessible. Under such circumstances, it becomes crucial that the integrated view be *self-maintainable*, meaning that view maintenance should be possible without requiring access to any underlying database, and without access to any information beyond the view itself and the log of the changes.

The integrated materialized view is used to provide fast query answering over the distributed databases. However, the view needs to be changed as the component databases change. The view may be difficult to maintain because not all the underlying databases may be available always, or the databases may be expensive to access. In addition, a distributed update involves concurrency control problems [ZGMHW95], interface translations, `etc`. Therefore, it is desirable that the view be maintained using solely the contents of the view and the modification. Multiple remote database accesses, and all its associated problems, can be avoided if such maintenance is possible.

Whether a view can be so maintained depends on both the definition of the view and on the type of modification. If for a given view and modification we can indeed determine the effect of the modification using only the contents of the view, then we say that the view is *self-maintainable* for that modification.

When integrating data, say about customers, from several databases, it is important to keep data about a customer who exists in only some of the databases. The outer-join operation in relational systems is useful when integrating databases in this fashion.

[1]In most data integration situations, autonomy of the underlying databases requires that updates to these databases be applied directly. As such, the problem of updating underlying databases through a view is also important in customer care systems, and we discuss it separately.

Materialized views are used for data integration in [ZHKF95, GJM94]. Objects that reside in multiple databases are integrated to give a larger object if the child objects "match." Matching for relational tuples using outer-joins and a *match* operator is done in [GJM94], while more general matching conditions are discussed in [ZHKF95]. The matching conditions of [ZHKF95] may be expensive, but by materializing the composed objects, in part or fully, the objects can be used inexpensively.

[ZGMHW95] also develop a warehousing model using materialized views. They present a concurrency control algorithm to ensure that queries to remote databases are correctly handled while doing view maintenance at the warehouse. The concurrency control problem they solve only appears if view maintenance requires access to base relations. [LMSS95] present a very different model of integration. They consider views defined using some remote and some local relations. They materialize the view only partially, without accessing the remote relation, by retaining a reference to the remote relation as a constraint in the view tuples. This model requires access to the remote databases during queries and is thus different from a warehousing model.

8.2 View Maintenance

We have developed several technqiues for maintenance of views, including the counting algorithm [GMS93] for maintenance of SQL views, an algorithm for maintaining outer-join views, self-maintenance [GJM94], and chronicle views [JMS95].

View Maintenance by Counting

The counting Algorithm [GMS93] applies to SQL views that may or may not have duplicates, and that may be defined using UNION, negation, and aggregation. The basic idea in the counting algorithm is to keep a count of the number of derivations for each view tuple as extra information in the view. We illustrate the counting algorithm using an example.

EXAMPLE 8.1 Let link(S, D) be a base relation where a tuple link(a, b) means that there is a link from source node a to destination node b. Consider a view hop where hop(c, d) is true if c is connected to d via two links *i.e.* there is a link from node c to some node x and a link from this node x to node d. The view definition for hop is written in SQL as:

```
CREATE VIEW hop(S, D) AS
       (SELECT DISTINCT l1.S, l2.D FROM link l1, link l2 WHERE l1.D = l2.S)
```

Given link $= \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, the view hop evaluates to $\{(a, c), (a, e)\}$. The tuple hop(a, e) has a unique derivation. hop(a, c) on the other hand has two derivations. If the view had duplicate semantics (did not have the DISTINCT operator) then hop(a, e) would have a count of 1 and hop(a, c) would have a count of 2. The counting algorithm pretends that the view has duplicate semantics, and stores these counts.

Suppose the tuple link(a, b) is deleted. Then we can see that hop can be recomputed as $\{(a, c)\}$. The counting algorithm infers that one derivation of each of the tuples hop(a, c) and hop(a, e) is deleted. The algorithm uses the stored counts to infer that hop(a, c) has one remaining derivation and therefore only deletes hop(a, e), which has no remaining derivation. ■

Self-Maintenance

Views that can be maintained using only the materialized view and key constraints are called *self-maintainable* views in [GJM94].

Definition 8.1 (Self Maintainability With Respect to a Modification Type): A view V is said to be self-maintainable with respect to a modification type (insertion, deletion, or update) to a base relation R if for all database states, the view can be self-maintained in response to all instances of a modification of the indicated type to the base relation R. ■

[GJM94] shows that most SPJ views are not self-maintainable with respect to insertions, but they are often self-maintainable with respect to deletions and updates. Conditions for determining self-maintainability are given. Further, it is shown that it helps to maintain a view if we model an update independently and not as a deletion followed by an insertion. It is also shown that outer-join views can often be self-maintained. This is particularly useful in data integration, since outer-joins are likely to be used heavily in doing data integration.

8.2.1 Chronicle Views

Banking, retailing, and billing systems deal with a continuous stream of transactional data generated by the custoemr using services. This ordered sequence of transactional tuples has been called a chronicle [JMS95]. One characteristic of a chronicle is that it can get very large, and it can be beyond the capacity of any database system to even store, far less access, for answering queries. Materialized views provide a way to answer queries over the chronicle without accessing the chronicle.

Materialized views can be defined to compute and store summaries of interest over the chronicles (the balance for each customer in a banking system, or the profits of each store in the retailing system). View maintenance techniques are needed to maintain these summaries as new transactions are added to the chronicle, but without accessing the old entries in the chronicle [JMS95].

8.3 View Updates

We would like to relieve a CSA from having to execute multiple transactions on component databases. Instead, a CSA would execute a single global update operation that would be translated into a set of transactions on individual databases by the system. In addition, since the system maintains the materialized view, this update must be reflected in the view as well. Instead of waiting for the normal view maintenance mechanisms to update the view after the individual transactions modify the data on component databases, it is desirable for efficiency to update the view directly.

Updating the view directly poses interesting concurrency control problems. Indeed, a view update at the data warehouse is translated into a set of update transactions at each database. No concurrency control between these update transactions is done to ensure that these are executed as a part of a global transaction.

Simultaneously, updates occur directly at the individual databases, these are transmitted to the warehouse, and these trigger view maintenance processes. Maintenance of these views causes queries to the component databases. How can the system guarantee consistency of the maintenance operation in presence of yet uncompleted updates?

8.4 Data currency

A specific requirement for the customer care system (as opposed to a traditional datawarehouse purely for decision support) is that the requirements for the currency of the data in the view are much more stringent. Different types of data may require different level of currency. Thus, the view definition language must provide the means for specifying the currency levels.

9 Workflow and Sagas

Workflow and extended transaction management technology is going to play an important part in Welcome. To illustrate the idea of a workflow transaction, consider the following scenario from the Ordering and Service Delivery.

A customer orders a complex service. This order (called a global transaction) includes four tasks:

1. Purchasing some hardware (denoted Hardware)
2. Purchasing a database license from Oracle (denoted Oracle)
3. Provisioning long distance service (denoted Provisioning)
4. Generating a single bill (denoted Billing)

Some of these tasks may in turn consist of component tasks in the hierarchical manner. Assume, for instance, that hardware can be purchased internally (from GIS) by executing Hardware1 task, or by executing a more expensive Hardware2 task that purchases the hardware from an external provider. (We may have to resort to Hardware2 if Hardware1 aborts, e.g., because GIS.)

Thus, the global transaction consists of a set of semi-independent tasks, some of which are legacy or even third-party applications. The execution of these tasks is subject to the following constraints:

1. Billing task can start only after the Hardware and Oracle tasks complete (it needs bills generated by these tasks to produce a single bill).
2. Within the Hardware task, Hardware2 subtask should run only if Hardware1 aborts.
3. In any of the four top-level tasks that comprise the global transaction abort (e.g., Billing may abort due to credit problems with the customer), all other tasks must either abort or be compensated.

Such applications can be implemented using workflow transaction management technology (see, e.g., [GHKM94, CR92]). At the heart of a workflow management system is a workflow monitor. When a new integrated offer is designed, the tasks that comprise the offer (including compensating tasks) are made known to the monitor. Also, restrictions and dependencies between tasks are specified. (Different ways to specify the dependencies have been proposed).

Once a global transaction is initiated (a customer orders the service), the monitor schedules and coordinates the execution of the component tasks. Note that the monitor may engage human agents in some stages of the execution of the global task. For instance, to order supplies from a third party (to run the task that purchases Oracle in our example), the monitor may run a simple task that sends mail to a human operator. When the operator completes the purchase (say, upon receiving the bill from Oracle), (s)he would enter the relevant information into a computer, thus letting the monitor know that the corresponding task has committed.

A specific case of workflow transactions is "sagas" [GMS87]. A saga is a long-running transaction that consists of a set of individual subtransactions. Individual subtransactions exhibit full transactional semantics. In other words, they are executed atomically. However, a global transaction may not execute as a single atomic action: another global transaction may see the effect of some but not all subtransactions of a global transaction.

The guarantee the saga model provides is that a global transaction either executes completely, or all its subtransactions will be aborted or compensated. Thus, this model implements a specific type of dependency between subtransactions. By fixing the kind of dependency that subtransactions may have, the saga model eliminates the need for explicit specification of task dependencies.

ConTracts [RW91] are like Sagas but with a partial order specified for the execution of the component transactions.

10 Direct Customer Access to the Customer Care Platform

We would like to provide direct automated customer access to the customer care system to initiate service, check order status, billing information, etc. via telephone (speech recognition) or a customer computer system. Beside being a valuable capability for many customers, this would significantl reduce the number of CSAs needed. With regards to direct customer access, we would like to explore the following:

- Direct customer access to the customer care system via standard World Wide Web browsers such as Netscape and Mosaic. In a customer care system such an interface could allow a customer direct access to offer-related information (in the form of database views, mentioned above), eliminating the need for a customer service agent (CSA) to serve as an intermediary. Direct access by customers to information is a primary goal of the Welcome Customer Care architecture being developed by the Database Systems Research Department.

- Customer Care via cooperative computing techniques on the World Wide Web using standard browsers. This will allow the customer and the CSA to view and manipulate the same information in a synchronous manner. The customer and the CSA could simultaneously view the customer's offer-related information. This would eliminate much of need for verbal relay of information (as is mandated in current systems) between the customer and CSA. A visual interface would also undoubtably be more informative to a customer than a verbal exchange only when discussing the customer's data.

- Security and authentication issues due to direct customer access and cooperative browsing.

11 Customer Care System as a Data Warehouses

Most data warehouses are planned as vehicles for decision support. Intended applications can live with information that is not current.

Customer care warehouses, on the other hand, are intended for more than decision support. They require current customer information. At the very least, some part of the information must be current. This would allow them to respond to customer inquiries. Customer care systems must provide a single point of contact to the customer. The ability to update the views in the datawarehouse is important from the single point of contact requirement perspective.

Customer Care = DataWarehousing + Single point of Contact

Customer Care = DataWarehousing + WorkFlow + View Updates

References

[CR92] P. Chrysanthis and K. Ramamritham. Acta: The saga continues. In A. Elmagarmid, editor, *Advanced Transaction Models for New Applications.* Morgan-Kaufman, 1992.

[CS95] Michael Carey and Donovan Schneider, editors. *Proceedings of ACM SIGMOD 1995 International Conference on Management of Data*, San Jose, CA, May 23-25 1995.

[GHKM94] D. Georgakopoulos, M. Hornick, P. Krychniak, and F. Manola. Specification and management of extended transactions in a programmable transaction environment. In *Int. Conf. on Data Engineering*, pages 462–473, 1994.

[GJM94] Ashish Gupta, H. V. Jagadish, and Inderpal Singh Mumick. Data integration using self-maintainable views. Technical Memorandum 113880-941101-32, AT&T Bell Laboratories, November 1994.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In Umeshwar Dayal and Irv Traiger, editors, *Proceedings of ACM SIGMOD 1987 International Conference on Management of Data*, pages 249–259, San Francisco, CA, May 27-29 1987.

[GMS93] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of ACM SIGMOD 1993 International Conference on Management of Data*, Washington, DC, May 26-28 1993.

[JMS95] H. V. Jagadish, Inderpal Singh Mumick, and Avi Silberschatz. View maintenance issues in the chronicle data model. In *Proceedings of the Fourteenth Symposium on Principles of Database Systems (PODS)*, pages 113–124, San Jose, CA, May 22-24 1995.

[LMSS95] J. Lu, G. Moerkotte, J. Schu, and V. S. Subrahmanian. Efficient maintenance of materialized mediated views. In Carey and Schneider [CS95].

[RW91] Reuter and Wachter. The contract model. In *Quarterly Bulleting of IEEE Computer Society Technical Communication on Data Engineering.* IEEE, 1991.

[ZGMHW95] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View maintenance in a warehousing environment. In Carey and Schneider [CS95].

[ZHKF95] G. Zhou, R. Hull, R. King, and J-C. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of* 3^{rd} *Intl. Conf. on Cooperative Info. Sys*, pages 4–18, 1995.