

The 1C:Enterprise Platform as a Product of Engineering Thought

View from the Other Side

Today, practically every other product of human activity offered to the buyer is related to high technology. And that means that it not only satisfies a person's specific needs, but also embodies scientific and engineering thought. Obviously, of course, there are scientific advances that completely define the content and consumer value of specific products.

As an aside, we can recall the revolution in the field of cold-weather sleeping mats when the Izhevsk Plant brought out polyurethane mats in 1987. Of course no one in the tourist community can imagine sleeping on the heavy inflatable mattresses or the light, but awfully hard foam plastic bars anymore. But if you think about it, the newer polyurethane mats are an ingenious scientific thought in pure form: a piece of material that doesn't get wet, weighs practically nothing, and doesn't pass cold to a sleeping traveler's body even if he's lying on snow. Of course, the inventors of this product want to tell the buyer, "look what interesting and beautiful ideas we've put into our product, no one has ever found such a solution, they are very elegant and promising, only they have permitted us to make such a useful product." But naturally, you don't normally talk to a buyer that way. And of course that's right, because the buyer doesn't care how interesting it was for scientists and engineers developing the product or how highly their advances are rated by other scientists and engineers; he only cares how well it will meet his own very specific needs. The only exception might be entertaining items sold in various technical museums and novelty stores. Of course, there are popular science journals and films in which interested buyers can raise the curtain and satisfy their curiosity about the designer's train of thought. Certainly business software is not a product whose primary purpose is to demonstrate the development of scientific and technical thought. The system's main objective is to meet the user's most urgent (today's) and future (tomorrow's) needs. So most of the informational materials on 1C:Enterprise are aimed at describing these necessary and useful capabilities. However, in this article we want to go in a different direction and explain how the interesting ideas and solutions (from our view, naturally) are incorporated into the system, and of course to show what specific useful properties result from these ideas.

Before beginning our exposition proper, we need to make a short comment on terminology. This article is not aimed solely at people working with 1C:Enterprise. We think it could be of interest to many specialists interested in the development of economic and corporate application technologies regardless of which specific technologies and products they use. So we will minimize the use of terminology specific to 1C:Enterprise and mostly favor terminology that is in general use. Since no settled terminology yet exists for many of the issues addressed here, we will sometimes use different terms for similar concepts, and provide equivalents and explanations.

A small clarification is needed on the term *1C:Enterprise* itself and what part of it we are discussing here. 1C:Enterprise is a *system of programs*, including a platform and a set of business applications (application solutions) built on it, designed for various areas of activity and enterprise scales. In this article, we will speak mainly of the platform on which the business applications are built.

What We're Fighting for

First we want to briefly define what we consider the principal aim of developing technologies such as the 1C:Enterprise platform. If we look at the history of progress in computers and programming, we can note that despite such obvious areas of development as improving productivity, increasing the volume of data that can be processed, improving ergonomics, etc., we can trace a fairly clear effort along the entire path of progress *to raise the level of abstraction* of software systems. We can note that this striving to increase the level of abstraction is a quality practically unique to the field of computing, while in other areas of human activity the principal aims are more utilitarian. This might be the basic super-idea of all progress in computing because it aims, ultimately, at realizing the still-unattained idea of creating artificial intelligence, which was the dream of the very first generation of computer specialists. It's very easy to follow the history of the rising level of abstraction, from programming at the level of jumpers, to machine codes, to assembler, to structured programming languages, etc. Each phase represented an increase in the level of abstraction of the interaction between man (both user and developer) and computer. And so the basic objective of the 1C:Enterprise platform is primarily to increase the level of abstraction still further in the development and use of business applications. Naturally, this also involves solving traditional problems related to productivity, ergonomics, functionality, etc. But it is increasing the level of abstraction that permits a transition in development from more hardware-oriented, low-level concepts to more object-oriented and high-level ones, which means raising the level of the system's concepts to the level of users' and specialists' concepts in the subject area. Ultimately, this will enable considerable acceleration and unification of both system development and support, and this in turn will permit developers to achieve the desired effect substantially faster and to take account of the numerous specific and individual needs of users, that is, those for whom we really create information systems.

The Platform and Business Applications

Perhaps the most conceptual (fundamental) solution in the 1C:Enterprise platform is the existence of the platform itself and the concept of a business application.

In general, any developer of serious business software has a fairly powerful toolbox and set of technologies that he uses to build his solutions. However, when developing 1C:Enterprise we initially set out to develop a fully functional, integrated platform that would be used by many developer firms to develop the most varied business applications. Accordingly, we initially created the platform as a reproducible product that included all necessary technologies to support business applications, as well as the tools to develop and modify them.

In 1C:Enterprise, we established a clear division between the platform and the business applications. The platform is primarily the framework in which the business application functions. We could not find a precise Russian translation of "framework." It can be regarded as a basis, but it can also mean the environment in which the application runs. In addition, the platform naturally contains a set of tools needed for development, administration, and support of business applications. A business application is an independent entity, and can function as a separate software product, but it is entirely based on the technologies of the platform.

Note that the concept of a platform and platform-oriented software development is now generally accepted and is treated much more broadly than work in a certain operating system. "Platform" is understood to mean a runtime environment and a set of technologies at a certain level used as the basis for building a certain range of applications. So applications are actually based on several platforms lying at different levels. It is important that the platform provide the developer a certain model, generally one that largely isolates him from the concepts and details of the lower-level technologies and platforms on which it runs. 1C:Enterprise, as a high-level platform, does isolate the developer as much as possible from the technologies that it uses. This enables it to use various lower-level technologies. For example, 1C:Enterprise can use a

proprietary file engine as a database for lighter applications (standalone and small workgroups) and MS SQL Server to run in large workgroups and at the enterprise scale.

Perhaps the key feature of the 1C:Enterprise platform is the adequacy of its tools for carrying out the tasks performed by business applications. This ensures good coordination of all the technologies and tools used by the developer. The very presence of fairly noticeable “seams” between the various technologies used can often create major difficulties in development. A very simple example is the type system. On the 1C:Enterprise platform, the developer uses a single system of data types for working with the database and for implementing business logic and for creating interface solutions, so he has no type conversion problems at the junctures between these levels.

Another very important point is standardization. The existence of a single platform for a large number of applications permits creation of a single “culture layer” that includes people (developers, analysts, and user) and methodology (standard data types, algorithms, interface solutions). By relying on this “culture layer,” the developer minimizes the effort wasted finding the necessary solution in any situation, from adding a new specialist to a project to implementing some subsystem of a business application according to the standard methodology.

One substantial advantage of the clear division into platform and business application is the high level of customizability of solutions to the client’s requirements. The platform’s development tools are aimed equally at the development of solutions and at their customization to a particular deployment. On the one hand, the development tools themselves provide solutions aimed specifically at the possibility of efficient application modification during deployment, and on the other hand, the very architecture used to build the application (and we will demonstrate this clearly below) makes customization possible. It is worth noting that development tools are supplied with practically every 1C:Enterprise kit and are used both for minor changes such as the print template and for substantial modification of applications, including the data structure and business logic.

The separation of the business application as an independent entity has actually permitted the establishment of an industry for the development, distribution, and support of the most varied business applications, focusing on the specifics of business tasks and abstracted from most of the technological details and fine points. For example, a company that has specialists, experience, and a reputation in a specific business can now quite realistically enter the business applications market quickly with its own product without wasting the several years it usually takes to develop a technology base for the solution.

Metadata: A Way of Describing a Business Application

In 1C:Enterprise, an application is not literally “written” in a programming language. A programming language is used, of course, but only where really necessary.

A business application is based on metadata, a structured declarative description of the business application. The metadata form a hierarchy of objects that describe all the business application’s components and all aspects of its behavior. When the business application runs, the platform actually “plays” the metadata, providing all the required functionality.

Metadata describe the data structure, type list, relations among various objects, various behavioral features of objects, aspects of the visual display of objects, the access rights restriction system, the application interface, etc. The metadata actually store information not only about “what to store in the database,” but also about “why” various pieces of information are stored and their roles in the system, and are interrelated as data files. The use of a programming language is limited mainly to performance of the business logic tasks that require algorithmic description, for example, the calculation of a tax.

What is the advantage of this approach to the description of a business application? First, the metadata description maximizes the use of visual editing; this converts most of the work of development from laborious coding to visual design. However, this is only the part of the

advantages that is visible at first glance. By describing an application in terms of metadata, the developer “communicates” to the platform a wealth of very useful information that the system can use effectively for the most diverse purposes. Based on the metadata description, the system automatically “builds” most of the mechanisms and objects that support the application’s function. For example, the metadata description is enough for the platform to automatically generate a user interface that supports the entry and editing of interrelated data. Another example is that it enables even an end user to create fairly complex reports without programming. When we describe these and other mechanisms below, we will show the benefits of using metadata in more detail.

Thus, the ideology of using metadata can be characterized by the phrase, “Let’s not program all functions of the solution being developed, let’s tell the platform the makeup, features, and relationships of the various parts of the solution and let the platform do as much as possible itself.”

It’s worth noting that the ideology of using metadata (metadata-driven technology) is now becoming more and more prevalent in the leading companies’ contemporary and future developments.

Building an Application from a Model

1C:Enterprise was originally based on a strict orientation toward building an application based on a certain model. This direction is very promising. Ideas of building business applications from models, for example, has been embodied in the OMG consortium’s MDA (model-driven architecture).

By “model,” we mean the entire ideology of building an application. This includes methods of building the data structure, types of relationships among data, principles for manipulation of data, ways of describing the business logic, ways of linking data with interface objects, the division of functionality among levels, and much more.

It is important that all business applications strictly adhere to the model, thereby ensuring that their behavior is consistent and predictable. The developer, in performing the specific task of reflecting a given area of the subject area in the application, actually has a fairly clear and limited set of ways of carrying out that task using the platform’s tools. On the one hand, this approach restricts (quite intentionally) the developer’s freedom, but on the other, it protects him from design errors and substantially increases the likelihood of quickly producing an operable solution that can later be elaborated and supported by either this or, if necessary, another developer.

An obvious consequence of this approach is that it maximizes the complete isolation of the business application’s developer from the details of the data storage technologies used, the organization of three-level architecture, etc. For example, as we have already noted above, all applications in 1C:Enterprise run as is both on a version of the system with a proprietary database file engine and on a version using MS SQL Server as a database server. The system creates and modifies the necessary data structures completely automatically based on the metadata description, and the developer is not buried in the details of storage formats of particular DBMSs. Similarly, the application’s manipulation of data is described in a high-level model provided by the system, and its execution automatically accounts for the features of the data repository used.

In our judgment, providing the developer a specific model abstracted from the specific assets used will come to dominate in the foreseeable future in modern development environments.

The presence of a single model crucially affects the simplicity of system development. All development is carried on in a single thorough system of concepts, in a single data type space, and it eliminates the need to develop several models of representation or waste effort on the transitions between them at various levels.

For example, the description of objects or entities in metadata immediately defines the built-in language objects and the database structures required to store them. All later manipulations of

these objects, whether in RAM or in the database, are performed consistently, without the difficulties of overcoming the “barriers” of different notations usually encountered when direct interaction with the DBMS and all-purpose programming languages are used.

Data Management

One key respect in which all the corporate and business application development tools differ is the paradigm used to manipulate data. Obviously for these tasks, working with data is the basic content, and also the most problematic issue as well. This is because in many areas software products have reached a level “close to perfection” (for example, in text editors), but economic software, to put it mildly, has a long way to go. Economic systems are constantly “under the gun” of contradictory requirements—to process the maximum possible volume of data with the maximum functionality in the minimum time. That is, the volumes of data are growing, the requirements for performing varied tasks are growing, and the requirements for scalability are being raised. Then add to this the demands to improve the ease of development, the system ergonomics, the capability for updating and modifying the system, and so forth. Improvements in perfection don’t seem to be anticipated in the foreseeable future. So all innovative technologies are always compromise solutions aimed at improving measures of some of these criteria. It goes without saying that these contradictions apply to all aspects of the system, but they affect data manipulation methods most obviously.

In modern world practice, there are several paradigms that are used in various combinations in business application development tools. 1C:Enterprise uses a mixed approach that has much in common with the approaches of promising developments from other companies, but at the same time also differs substantially from them.

1C:Enterprise uses object technology for all data modification operations (creation, editing, and deletion). That is, the developer controls the content of data storage spreadsheets using objects corresponding to the entities stored in the database. This enables the developer to realize handlers of events related to data modification. The system provides efficient technological support for the object technology, for example, object caching, maintenance of object and reference integrity, etc. Data can be read using either the object technology or a declarative query language, which is based on classical SQL but has a series of important extensions. These extensions are aimed both at supporting the handling of objects stored in the database and also at effectively solving economic problems. Below we will examine the query language in more detail.

One key advantage of this object-relation paradigm of data management is the clarity and simplicity of application development. The object technology, used mainly for data modification, makes business logic algorithms very readable, substantially reduces the number of errors in development and also ensures a high level of data integrity. In addition, a very important aspect of the object technology is that it offers the possibility of an organic transition to distributed and integrated systems, by supporting strict logical granularity of data manipulation. The declarative query language, in turn, makes it possible to obtain any arbitrary number of complex data samples and offers powerful aggregation capabilities needed to solve analytical problems. In our judgment, this type of mixed technology will be used in one form or another in developments in the foreseeable future.

Here, jumping ahead a little, we should say that the 1C:Enterprise model realizes the most modern concept of information handling, unifying three methods of data representation: storage of entities in a database, representation of entities as objects in a programming language, and representation of entities in XML format. In fact, any information, depending on the current operating mode, can be represented by one of these three methods.

Entity persistence is accomplished in the database, which provides storage and efficient processing of large volumes of data. For modification operations, data are converted to objects in a built-in language. When exchanged in the distributed infobase or with other information

systems, entities are converted to XML format. It is important here that all three methods of representation rely on a single system of concepts, and the developer need make no effort to transform information from one representation to another.

The fact that the realization of this approach on the 1C:Enterprise platform agrees well with the direction of evolution of corporate systems is not just important from the standpoint of assessing the platform's advancement. It permits 1C:Enterprise to be integrated into future heterogeneous solutions, which in our judgment (and many international experts agree) will gradually become dominant.

One distinctive feature of the 1C:Enterprise model that in our judgment has no direct equivalent in other systems is the division of all data stored in the database into object data and nonobject data. Note that object access is used for data manipulation with both types of data. Here we are using the word "object" in different senses—an object stored in a database, and an object used to manipulate data.

This division corresponds to the actual nature of the data. The subject area always contains object-type entities, such as "clients," "individuals," and "products." In these, an object has a certain "identity" that is independent of the data that describe it. On the other hand, there are entities whose records do not have an object nature, such as a record of the arrival of a specific product in a particular warehouse is only information on the movement of the product and has no "identity" other than the information stored in the record. For example, changing the product completely alters the meaning of the record of its movement. That is, the record is meaningless unless the specific values of the fields are given.

This semantic distinction among entities has fairly many manifestations and peculiarities in data manipulation. The 1C:Enterprise platform, by sharing the methods of handling these types of entities, actually offers the developer a finished methodology that permits him to perform various tasks closely related to the nature of those entities in a natural fashion.

The handling of object entities is supported by representing database objects as objects of the corresponding type in the built-in language, and also by the use of separate types to represent object references (references to database objects). It is always easy to obtain a reference from an object and to obtain the object from the database using a reference. This technology provides a clear and natural way of describing business logic algorithms that manipulate objects in source code, and also ensures the logical integrity of data in all operations. The use of this technology for manipulating object data resembles the writing of applications in object databases, but data are saved in relational DBMS tables that are reflections of the object representation, and the modules of the built-in language can simultaneously contain several objects representing a single database object.

On the one hand, object handling looks fairly simple to the business application developer. On the other, the 1C:Enterprise platform provides powerful support for efficient object handling. The platform's mechanisms provide support for unique object identifiers (references), object versioning, and pessimistic and optimistic locking. Optimistic locking is used to ensure the logical integrity of object changes, while pessimistic locking permits control of the simultaneous editing of the same objects by different users in the 1C:Enterprise interface. The platform optimizes the reading of these objects by using a caching system for these objects, both within and without transactions. When objects are modified, "smart write" technology is implemented, the system follows changes to objects and actually writes only the changed data, thereby ensuring the integrity of the object's modification.

The handling of nonobject entities is supported by the use of special objects: record sets that essentially consist of collections of records. Record sets support data reading and modification with the requisite granularity, thereby providing the required application logic. This technology has no direct equivalent in other systems. We think it is a fairly effective and convenient way of combining relational logic and the object method of manipulating data for nonobject data.

It is important that both technologies used support a simple and natural business logic algorithm writing style in the source codes of the business application. As he reads this kind of algorithm, the developer quickly understands the task being performed and can easily debug and correct the source code. This makes the development and support process much more efficient than in systems where interaction with the database requires complex transformations to be run in the code, with the transfer of data from object structures to structures designed to be written to the database and back.

Another important feature of the object technology of the 1C:Enterprise platform is that the very same objects that represent data in modules of the built-in language (for both object and nonobject data) are also used to display data in the interface. Form controls are directly linked to objects that represent data, and support their viewing and editing by the user without any additional effort on the developer's part.

For object data, the 1C:Enterprise platform supports a mechanism of representations. It is responsible for displaying values that represent references to database objects in the interface. Based on the metadata attributes, the system automatically generates a representation when it needs to display a value, optimizing data retrieval where possible using a special cache and other mechanisms. The query mechanism and the report-building mechanisms also provide special technology for working with representations, which permit reference field representations to be obtained universally if a request to display data in the user interface is generated, and the display of representations to be automatically included in reports for fields containing reference values. It is important that the representation mechanism provide the developer simple and natural handling of object references in the organization of the user interface, in the debugging of business logic algorithms, and in other cases, but minimize database accesses.

In addition to the data manipulation objects and query mechanism described, the system offers another specific way of handling data: dynamic samples. This mechanism allows the developer to work with large volumes of data, supporting block reads of information. The developer indicates only which data he requires, and the system automatically accesses the database at the required granularity. It is important that no specific means of dynamically reading a specific DBMS, which would require holding an open sample in memory, are used; rather, requests for successively sampled record blocks are generated automatically. This mechanism is also used to process data in modules and for visual inspection of long lists.

Another important data handling solution in 1C:Enterprise is support for compound data types in table fields. This possibility, in our judgment, has no near equivalent in any other system. When describing the field type of some object, the developer can select not just one of the available types, but practically any combination of types (with minor constraints). For example, the single Payer field in a document showing a transaction with a bank can hold a reference to a legal entity or an individual depending on the particular transaction. But this is a fairly simple example. The possibility of working with compound types permits the performance of tasks such as the storage of arbitrary characteristics of products, the maintenance of analytical accounts based on bookkeeping records for an arbitrary set of user-customizable analytical sections, etc. It is important that the system does more than just permit storage of dissimilar values. All the system's mechanisms fully support this possibility so that the developer is freed from the substantial additional costs of including compound-type fields in his application. Primarily, we must note the full support provided by the database mechanism and query language for working with compound-type fields. For these fields, it supports the entire set of required actions, such as sorting, comparison, aggregation, etc. Another important point is the support for compound types in the system's interface tools. For example, when working with such a compound type, an input field control provides a full set of capabilities for editing values such as type selection, editing of values of all types making up the field, restriction of selectable types if required in the particular case, multiple selection if several values must be input, etc.

The 1C:Enterprise query mechanism also deserves separate attention. We have already mentioned that it is based on standard SQL structures, but has several substantial extensions.

First, we should note the query language's support for the object nature of data stored in the database. All query language operators support the handling of reference types (fields that store references to database objects). For example, "point" access to fields is supported without limitation on the number of levels (a sample of a field such as "Account.Product.Manufacturer.Country" can be entered). Database objects permit access to nested tables, both to separate tables and to ordinary object fields containing record sets.

Another important capability is the standard implementation of retrieval of multidimensional totals with arbitrary measurement tabbing order. The system supports possibilities such as the combination of multidimensional measurement tabbing and multilevel tabbing of the hierarchy of values of each measurement (for example, multilevel organizational structure or grouping of products), special modes for processing results by period, selection by an object that has subordinate objects, "clean" results and results that include subordinate objects, etc.

The query mechanism supports automatic data sorting mode based on application object attributes defined in the metadata. This enables the developer to use a query to obtain a report or visualize data without naming specific sort fields, but to obtain the standard sort for the data used by activating the autosort mode.

Another powerful tool of the query mechanism is virtual tables. They support access to arbitrary data represented by various application mechanisms without the requirement to write complex queries. For example, a virtual table can be accessed to obtain product balances in a cross-section of warehouses and product lines at a given moment in time. We should note that this mechanism is not the simple storage of standard queries (view). When using virtual tables, the developer specifies the set of parameters that describe the required sample so that the virtual table's operating logic will vary substantially according to the current application. The input parameters are not only specific values, but also, for example, complex conditions. On the one hand, the application developer works with the virtual table practically the same way as with an ordinary one. On the other hand, the system generates a database query so as to maximize the efficiency of information retrieval; for example, saved intermediate results can be used when accessing accounting mechanism data.

Standard Prototypes of Application Objects

If we compare various models and tools for developing business applications, perhaps the most essential characteristic that distinguishes them is the terms (we could even say the "paradigm") in which the business solution is described. Naturally, several methods of description can be used in each development tool, but one set of concepts is always fundamental.

As examples of existing approaches, we can cite the description of solutions in terms of relational tables and entity-persistent classes of the object programming language.

The 1C:Enterprise development model uses an approach to which we have found no clear equivalent in other systems. The entire application is described in metadata as a set of application objects that relate to a strictly defined set of prototypes (classes). From the standpoint of alternatives in existing terminology, we can also call the created objects "business components," and their prototypes "patterns." Each application object prototype is responsible for showing a certain range of objects or subject-area processes in the application, which are distinguished by similar behavioral characteristics and similar roles in the overall picture. Examples of such prototypes are catalogs, documents, and accumulation registers. Each application object prototype has a predefined realization, which includes the most varied aspects that ensure the function of objects created from it: the set of persistent entities with some number of predefined fields, the set of programming language types with their required methods, attributes, and events, the realization of standard operations corresponding to the task being

performed, standard viewing and editing methods in the interface segment, standard methods of controlling access rights, etc.

When displaying subject-area objects in an application, the developer defines a suitable prototype and uses it to create a metadata object. Then he can adjust the various attributes that define the behavioral features of objects based on the prototype, augment the object's data structure if necessary, implement the needed set of methods, define event handlers, realize the object's specific display in the interface, etc. For simple objects, this predefinition may not be needed at all, and the object's entire development will amount to entering its name in the metadata attributes. For complex objects, the developer may define both the data structure and the business logic algorithms in some detail, but the standard functionality defined in the system will operate with no effort on the developer's part.

It is important that the number of these prototypes not be large, about 20. This enables developers to learn them well and make efficient use of the necessary set of tools provided by the system to perform any and all tasks in the subject area.

Thus, the 1C:Enterprise metadata structure is not simply a set of descriptions of objects expressed in definite standardized terms. The entire application actually consists of objects clearly classified according to the roles that they play in the business application. The existence of this approach substantially magnifies the benefits of both describing the system in metadata terms and building applications from a model.

The fact that metadata objects have standard functionality and predefined roles enables the system to support a much larger scope of functionality automatically, both directly in these objects and in common mechanisms that work with all objects. Actually, knowing a given object's purpose (role in the application), the platform can provide it with an appropriate "individual approach" in the most varied situations, and this will require no effort on the part of the application developer. For example, the standard report mechanism can automatically "adapt" to display various types of information most effectively.

On the other hand, the model's use of predefined classes standardizes the application building model. This standardization may not be very substantial for one solution taken by itself, but it has a far-reaching effect on the situation in creating an industry for developing and supporting numerous applications. Standardization helps substantially simplify design decision-making for the developer, and reduces the complexity and labor intensity of support several-fold. For example, when a new specialist must be added to a project's development, or transfer support and development of a solution to another specialist, the developer can literally in minutes, based on a cursory look at the metadata structure, get a general idea of the structure and basic functionality of the application because the metadata structure divides all application objects into roles generally accepted in the 1C:Enterprise model. Later he will also be able to quickly analyze the setup and business logic of selected subsystems. Experience shows that these measures differ substantially from those achieved by using other approaches to application description.

Application Mechanisms

Above we described the actual principle for building an application based on the use of standard application object prototypes. It obviously makes no sense in this article to describe all the available object prototypes in detail. But in this section, we will try to present some brief overview of prototypes and the capabilities they offer, in order to demonstrate the ideas incorporated in the existing business application building model.

Let us begin by giving examples of fairly simple prototypes: catalogs and documents. Catalogs describe the entities available in a given subject area: directories with more or less constant content. For example, they include a list of a company's products, a list of its clients,

etc. Catalogs help to support the hierarchy, to divide data into objects proper and groups, and also provide much more service corresponding to their purpose. Documents are responsible for entities that reflect business events occurring in the company's life. For example, these include the receipt of materials, payment of money through a bank, employee hiring, etc. This object prototype makes it possible to show business events in various accounting mechanisms, maintains control over the order of depiction of events, enables the thorough numbering of various types of objects, etc.

Let us dwell in a little more detail on several capabilities. As you know, the classical relational model has no standard tools for supporting data hierarchy, and realizing requires painstaking work. Catalogs (as well as some other application object prototypes) have the standard capability of supporting multilevel hierarchy. This mechanism is enabled simply by setting the corresponding attribute in the metadata. Hierarchy support immediately propagates to all aspects of the application object's use. For example, a prototype supports the necessary attributes and methods in an object data manipulation model (definition of an object's level, checking for hierarchy looping, etc.). The interface mechanisms provide data representation as a hierarchical list or tree, with support for navigation between levels and interactive modification of the hierarchy. The report mechanisms provide generation of hierarchical reports, as well as a multilevel hierarchy of results in all reports containing objects of this type as report groupings.

Another example of standard object capabilities is support for the document posting mechanism. This mechanism offers the developer a standard model of the linking of information on events occurring at the enterprise and various accounting mechanisms. Any information entered by the user in document form can be shown in any accounting mechanisms (management accounts, planning, account books, etc.). The developer need only create a link in the metadata attributes between the documents and the accounting mechanisms, and describe the document posting algorithm. The system will perform all necessary posting and cancellation actions automatically. Additional capabilities are also offered, such as support for displaying events in real time, support for the order of display of events occurring at the enterprise when they are modified retroactively, etc. It is important that the unified model of source data and accounting mechanisms offered this way does not just simplify development, it also provides all applications a unified model with predictable behavior that substantially eases the system's development and support.

The presence of such rather simple prototypes substantially simplifies development, but it would be worthwhile to dwell in more detail on the more complex prototypes of application objects.

1C:Enterprise object prototypes that form the periodic payment mechanism are very interesting. The periodic payment mechanism can be regarded as an all-purpose "engine" for performing payroll tasks of any complexity. In fact, payroll is the most typical application for this mechanism, but the mechanism itself is not oriented toward this task and can be used successfully to perform other tasks requiring description of periodic settlements with complex interrelationships, for example, the payment of dividends, utility bills, etc. The mechanism includes a set of standard strategies used to perform these tasks. For example, it supports the possibility of displacing some types of payments in favor of others when they overlap in time. As applied to the payroll task, this capability solves a whole series of problems related to accruals and withholdings that intersect in time, which are subject to a complex relationship of mutual exclusion in effective time. Another example is the possibility of establishing a relationship between various types of accruals and withholdings by effective period. Thus, the mechanism offers a definite mathematical model that establishes links between various types of payments and offers the developer data processing mechanisms that act on the basis of that model.

Without delving further into the details of this mechanism, it is worth noting that it permits a complete description of the payroll scheme based on supplied classes of application metadata

objects and their inherent capabilities. Using this mechanism as an example, we can demonstrate fairly clearly the benefit of using the proposed approach. To our knowledge, there does not currently exist any economic software development tool with a similar mechanism. In all the cases known to us, the payroll module is realized at the expense of fairly high development and implementation costs. And this module must be substantially or completely rewritten for each country because it is heavily linked to local laws. The use of the complex periodic payment mechanism in 1C:Enterprise enables the development of such modules to be shifted from design “from scratch” to implementation according to a standard scheme. In our judgment, this reduces costs by an order of magnitude, but most importantly, it substantially reduces the cost and time of changes required in the course of the system’s support.

Another example that we could cite is the characteristic mechanism. It is designed to perform tasks that fit poorly into the classical relational database structure. One such task, as a rule, is the creation of a structure to display the various attributes or characteristics of a product line. If a specific solution is being designed for one organization with a clear focus on a definite group of products, all the required attributes can be incorporated directly into the structure of the product catalog. For example, for an auto dealership, this would mean characteristics such as color, engine displacement, etc. But if a reproducible solution or a solution for an organization that produces or sells a diversified line of products is being developed, this option is inappropriate because the set of characteristics must obviously depend on the specific class of products and adding a new characteristic should not require reworking the application. Obviously, the implementation of such a solution fits poorly into a relational structure and requires a special approach. The characteristic mechanism implemented in 1C:Enterprise offers the developer a standard tool for performing these tasks. The specialized class of application objects is designed to describe characteristic types. Each characteristic type can have its own data type. For example, engine displacement will be specified as a number, while color will be selected from a catalog. It is important that specific characteristic types can be defined not only in the application itself, but can also be added by the user in the process of working with the system without altering the application. The application developer can enable the characteristic mechanism for various data and with various options; for example, he can realize the storage of logically related combinations of product characteristics (options) and the calculation of their cost across the available combinations of characteristics.

Another interesting solution is the work-flow mechanism. This mechanism enables the developer to organize user collaboration in the performance of standard sequences of organizational or commercial actions. Many existing information systems perform work-flow tasks using separate products integrated with the problems that perform economic tasks. On the 1C:Enterprise platform, the work-flow mechanism is fully integrated into the system so that neither the developer nor the user sees the “seam” dividing the mechanism from the other functionality. The work-flow mechanism includes tools for describing work-flow charts in the application, adjusting role routing, generating tasks for users at each waypoint, controlling the work flow en route, and linking the work flow to the application’s remaining functionality. The work-flow mechanism offers the developer flexible capabilities for branching the process and generating tasks. For example, in addition to ordinary branching on various conditions, the developer can visually describe the parallel passage of several branches on a path and specify join points. The system provides the possibility of sending one task to a group of potential performers, for example, if an invoice must be issued by one of a section’s managers. It also provides for the converse possibility of specifying several tasks at some point on a path, for example, if all section managers must submit financial reports. Role-oriented routing permits a task to be generated directly for a particular employee, and also permits tasks to be assigned by roles, organizational units, and other criteria that the application developer can describe. Users can specify the current assignment of employee duties taking account of temporary substitutes, combinations of several jobs, etc. It is important to note that the work-flow mechanism offers a

finished strategy for performing these tasks. To describe simple work flows, it is enough to visually describe the flow chart and specify the branching conditions in waypoint handlers. The system does all the rest automatically. When implementing complex work flows, the developer's efforts are mostly aimed at realizing the close linkage between the work flow and the application's functionality.

We have shown with these examples that each of the application object prototypes and the mechanisms that joint them covers a specific area of tasks in the subject area, substantially cutting the costs of development, and very importantly, standardizing applications. We would also like to note briefly several other mechanisms, in order to give an idea of the range of tools available to the application developer.

The accumulation register mechanism gives the developer a tool for performing tasks related to the accounting of cash and the movement of assets (tangible and monetary). The mechanism supports a multidimensional accounting system with an arbitrary set of measurements and resources, and supports optimization of totaling at various points in time. The mechanism enables efficient accomplishment of the tasks of material accounting, production and financial planning, settlement with vendors and customers, and so forth.

The bookkeeping mechanism is an all-purpose "engine" for automating bookkeeping in various accounting models. It can be used in various countries and is highly flexible and customizable. It is based on support for double-entry bookkeeping. It can handle financial transactions, including both the simple correspondences (one debit and one credit) accepted in Russian accounting and in the accounting models of some other countries, as well as the complex correspondences accepted in most Western countries (several debits and several credits). The mechanism supports a multidimensional accounting system with an arbitrary set of measurements and resources. A unique property of the mechanism is its ability to customize the set of secondary measurements independently for each account. The measurement set can be defined either by the developer in the application or by the user as he works with the system. The mechanism offers automatic multilevel and multidimensional totaling over a cross-section of periods, cross-correspondences, etc.

The information mechanism is designed to perform tasks related to the storage of diverse information on objects in various cross-sections. For example, data on prices and products can be stored across the product itself and the client category, etc. When necessary, the storage of an information change history can be supported, with generation of both actual data at the current moment and data from any previous point in time. The mechanism provides all the logic for handling stored information, from the data manipulation level to automatic contextual presentation of available information on various objects to the user in the system's interface.

In concluding this section, we would like to reiterate that in the 1C:Enterprise platform's development model, the makeup of application mechanisms and their component application object prototypes is not simply a set of supplied templates that the developer can use "as necessary." The entire application is based on the use of the existing set of mechanisms and their component application object prototypes. These sets have been worked out during platform development so as to cover all subject-area tasks with a completely visible set. Thus, when designing the system for each subject-area object, the developer defines a corresponding 1C:Enterprise application object prototype. The system actually uses the available set of prototypes to impose a standard design model on the developer, and in our judgment, this permits substantial reduction in cost for both development and support of the application.

The High-Level Interface Model

The question of the user interface organization is obviously of special importance for a business application. First, because for many users, these applications are their basic tools, and such users spend most of their time working with them, often without being professional

computer users. Second, the business application interface is usually very “large.” Business applications generally contain between several hundred and several thousand forms. In addition, business application forms have a tendency to change periodically, for example, as the system develops. Accordingly, the costs of developing an application interface are very high. For this reason, the 1C:Enterprise platform implements a fairly large set of mechanisms aimed at rapidly developing ergonomic user interfaces. The 1C:Enterprise interface implements a proprietary window model, a proprietary form system, a proprietary set of controls, and other mechanisms. Naturally, the scope of this article does not permit a detailed description of all of their capabilities. We will merely try to describe the general concept and illustrate its use with selected examples.

So the main objectives that must be met from our standpoint are speed of development and ease of use.

Our main idea in organizing the 1C:Enterprise interface was to maximize the use of information from metadata and data manipulation objects to support the interface’s automatic operation without detailed customization by the developer. We have already said that the built-in language objects used to manipulate data are also used to present data in the interface. The developer need only link a data manipulation object to a specific control and the interface mechanism will provide all data viewing and modification. The greatest benefit comes from the ability to create links between the data manipulation object and the form itself, not with the individual controls. Then the system will automatically provide all the form’s functionality related to data viewing and editing, as well as linking to other forms. To this end, 1C:Enterprise forms implement a mechanism of form extensions and control extensions. Necessary extensions are automatically “turned on” depending on the data type to which the control or form is linked. Extensions automatically provide practically required functionality for data entry, editing, and viewing. Thanks to the standard application object prototypes, the extensions account for the specifics of each type of application object, providing all needed service to the user, and also offer the developer tools for customization to the specifics of the particular application data type.

For example, an input field provides all necessary actions for editing reference values, such as the selection from special selection forms, selection of multiple values, resetting of values, highlighting and automatic selection of unfilled values, switching to editing forms for selected values, etc. A very effective tool is the string input mechanism. The metadata attributes specify the makeup of fields, from which users can quickly find selectable objects, e.g., code, item, description. When they enter several characters in the input field, the system automatically supplies the value of the one specified in the metadata fields that begins with the characters entered. If several such values are found, the user is prompted to choose a value from a list. This mechanism requires no effort on the part of the developer, because it relies on the metadata attributes, and accelerates mass data entry several times by eliminating the need to select values from list forms.

Another interesting example is the table field. This control consists of a powerful data display and editing mechanism that enables the user to view large lists with dynamic reading. We have already spoken above about the dynamic reading mechanism as a data manipulation tool. Here it permits data files of practically unlimited size to be viewed in forms without reading them all into memory. From the user’s standpoint, navigating through the list hardly differs from viewing data in the computer’s memory. The mechanism provides flexible search and filtering capabilities and the ability to view hierarchical data, edit data in lists and separate forms, customize the content of columns, sort, print lists, and output lists in various formats, etc.

For forms, lists, and other interface elements, the system automatically generates a command interface (buttons, menus, and whole command panels) that support all viewing and editing actions, as well as linking to other interface forms and various utility functions.

Having described a metadata object in the application, the developer can omit form creation entirely; all necessary forms will be created automatically as the system operates. If the developer creates a form in order to give it specific attributes, then a wizard generates a standard form that initially contains no code in the built-in language, because control and form extensions provide all the required functionality. Naturally, the developer can also insert controls into any form without the wizard, link them to data, and obtain a fully operational form without writing module procedures. Accordingly, the developer must write the source code in the form module only when he wants to redefine or extend the standard functionality.

It is important that the 1C:Enterprise interface does not just support the operation of separate forms and controls. The standard mechanics of application object form control provides various options for links between forms used in value selection, multiple selection and data detailing. The 1C:Enterprise platform actually offers the developer a finished strategy for organizing the entire business application interface, which has ways of implementing practically all necessary user work scenarios.

Through the use of metadata values on links between various objects, application object forms support the automatic generation of commands to switch between logically linked forms. So the user working with some document, without any effort on the developer's part, can gain access to records that reflect the document in various accounting mechanisms. In addition to links defined in the metadata by the application's business logic, the developer can also define additional links for switching between objects in the form of selection criteria that will also automatically be presented in the forms' command interface.

Another standard interface solution is the "input based on" mechanism. This enables the user to enter data in some objects by supplying all possible information automatically from other objects. For example, having written an invoice, the user can generate a waybill based on it by pressing a single key. It is important that the developer need only create the appropriate link between objects in the metadata and describe the filling of fields in the object, and the object form interface will automatically include this possibility.

The application developer is provided a standard tool for implementing a "replaceable" command interface. In applications with extensive functionality, this allows the user to switch to a different operating mode, with full or partial replacement of the command interface, without leaving the system.

The form mechanism provides a strategy for semantic identification of forms, permitting the user to avoid reopening already open forms, but instead to activate them when handling various information.

Thus, with the aid of form extensions and other mechanisms, an integrated user interface strategy is automatically supported, including various options for interaction between forms, automatic generation of a command interface for navigation among various system modes and functions, and all possible utility modes that make the user's work comfortable.

We have already noted that the 1C:Enterprise interface implements a fairly large number of special elements that do not use standard operating system tools. In creating the interface, we strove to realize a model most suitable to the specifics of the user's work with economic tasks.

The 1C:Enterprise implements a proprietary model of an application windowing system. It is oriented toward the user's work with a large number of heterogeneous information. For example, the user controls window maximization separately for each window, not as a single mode for the entire application (as done in standard MDI applications). In addition to traditional windows, docked and hidden windows are used. In the 1C:Enterprise interface, nonmodal windows can be called from a modal window, so that several layers of a multiwindow interface are actually formed. This permits the developer to open modally any application forms without altering their operating logic, because in modal windows, switches to any other windows are made the same as in ordinary ones.

The form mechanism supports automatic resizing of controls when the form is resized or the dividers in the form are moved. Thus for each form, the user can set the desired size for the entire form and for its parts, and the form will automatically use the entire space available to display the information efficiently. It is important that this mechanism operates without the need for any settings on the developer's part, but naturally it does permit fine tuning.

Forms and controls have a design aimed on the one hand at showing the maximum amount of information and on the other hand at reducing user boredom during a long work day. This is accomplished by using a "flat" interface design model close to Web design. The system provides a style mechanism to control interface format, which permits centralized changes to the application design. Practically every control is "equipped" with functionality aimed at the specifics of economic tasks. In addition, the system provides a series of controls directly aimed at analytical tasks, such as graphs, Gantt charts, etc.

The 1C:Enterprise interface mechanisms have many more interesting elements. In this article, we have listed only a few solutions. It is important that the full set of interface mechanisms is provided to the developer in fairly high-level terms. We have already stated that an important element of the development concept in 1C:Enterprise is the isolation of the application from various types of technical details. For example, the developer is not given the capability of controlling mouse movements or the detailed drawing of controls. Even fairly complex mechanisms are provided to the developer in such a form that they can work generally without any customization while at the same time permitting customization in simple high-level categories that require no special knowledge. An important result of this approach is that the applications support a modern fully functional interface, and only a small part of the source code relates to purely interface tasks, while most of it implements the actual business logic of the application interface.

To conclude the subject of the interface, it is worth mentioning the Web extension, the mechanism that makes possible a Web interface to the 1C:Enterprise applications. We will not relate the details of the technological setup of this mechanism. The most interesting aspect of its implementation, in our view, is that just like the main (rich) interface, it makes maximum use of information from metadata, as well as knowledge of the purpose and setup of application object prototypes for automatic generation of Web forms. For editing of any object or viewing of a list of objects, a corresponding form is created automatically by a Web extension if it is defined in the design. If the developer describes a form himself, he uses specialized controls that not only organize the link to data but also provide all necessary viewing and editing functionality. For example, the input fields support selection from drop-down lists and selection by first letter, the lists support hierarchical viewing and user customization of filtering and sorting. The system provides a large set of standard data "service" actions and independently organizes links between the application's forms. This ensures uniformity of the two interface versions for the user (naturally, with adjustment for the peculiarities of the Web interface), and also maximizes the development automation based on the use of metadata information.

Report Preparation Mechanisms

Economic software traditionally has two main components that define how the user works with the system. These are the interface, which is intended for data input, and mechanisms for obtaining reports. These are what is primarily responsible for presenting information for management decision-making. So report-generating mechanisms occupy a special place in 1C:Enterprise, too.

We think the implementation of report mechanisms in 1C:Enterprise contains a whole series of unique solutions with no direct equivalents in other systems. First of all, we should note that the 1C:Enterprise platform lacks the separate "report generation" tools traditional for most systems. To prepare reports, the developer is provided a whole series of mechanisms that can be

used in various combinations and in combination with other mechanisms. Besides the obvious benefit of flexibility for the developer, this gives a very important benefit when the user works with the system. Actually, the user doesn't see the boundaries between the general interface and the reporting mechanism as he works. We don't think a modern economic system should have any, so the necessary analytical information can be used in all modes and forms, and reports are mostly not generated for printing, but for interactive analysis. So in the 1C:Enterprise mechanisms, report preparation tools are organically integrated with other mechanisms and have powerful capabilities for interactive navigation by users.

One of the most interesting solutions among the mechanisms used to build reports is the report builder. This mechanism integrates several other mechanisms and enables the user to obtain the most functional report with the least effort.

The main idea of this mechanism is as follows. The developer or user specifies as briefly as possible only what information must be analyzed, and the builder does everything needed to produce a fully functional controllable report itself.

Source information sufficient for the report builder to work is specified using query code in the 1C:Enterprise query language. Because the finished virtual tables from accounting mechanisms can be used, the query code for a fairly complex report will take up only a few lines. And it need not always be written manually. Usually the code is generated by visual selection of available data in the query wizard (here we should note that the query wizard permits visual generation of queries of practically unlimited complexity, including combinations and nested queries, and is capable of "understanding" query code located anywhere in the module for visual editing).

Based on comprehensive analysis of the query code, the report builder independently generates the entire infrastructure needed both to produce a report and for all possible report settings by the user and navigation between reports. The user will be provided a form containing both the report proper and all possible tools for controlling it. The report itself may be in the form of a table with row and column hierarchy, but it can also be in the form of a graph, a pivot chart, or a pivot table. In addition, the form will include tools for controlling the content of fields included in the report, filtering data by complex compound criteria, placing groups in rows and columns of the report, and customizing the ordering of report data. Note that the interactive customization of filtering will include possibilities such as setting selection by data related to data in the report's reference fields with unlimited link depth, the ability to select the set of values by which information must be selected, and the ability to specify as values groups in whose hierarchy information must be selected. For example, the user can set the selection of manifests so that the report will include documents containing products whose attributes specify manufacturers belonging to the groups "Domestic" or "Foreign." In addition, the report builder includes all necessary information in the generated report form to support expansion of report cells (drill-down). When the user calls drill-down, the builder can automatically generate a more detailed report based on the information about the original report and data on the selected cell. In addition, the builder permits the user to save report settings, and thereby maintain a report history.

It is very important that all these capabilities are provided to the developer in finished form, but at the same time he can use them in various combinations and flexibly control the degrees of freedom offered to the user when working with a report. For example, he can specify in his query code what fields the user can use when filtering, sorting, grouping, etc. The report controls themselves are also included in the form at the developer's discretion. Since all parts of the mechanism can be used independently, the developer can use the builder to do more than prepare a classical report. For example, he can develop a group data modification mode in which all selection control mechanisms will be used, but instead of the report builder proper, a specific process will be executed on the selected information. Thus, the developer can include this mechanism in any interface solution in any combination.

Another important benefit of this mechanism, we think, is that the 1C:Enterprise applications use it to implement tools in which “power users” can actually create their own fairly complex reports.

Obviously, for presentation to the user, both the flexibility of report construction and the effectiveness of the display tools are important. 1C:Enterprise uses primarily spreadsheets for this purpose. The main idea of this solution is to use a spreadsheet model as a report formatting tool without using it as a computational model.

Unlike electronic spreadsheets, the 1C:Enterprise spreadsheet can work with very large table sizes, both vertically and horizontally. It contains a rich selection of display capabilities that enable the creation of well formatted reports as well as the creation of blank forms with precise object location by coordinates when necessary. An interesting distinction between the spreadsheet and standard electronic spreadsheets is the ability to describe rows with different column width settings. This is necessary in order to create complex reports including several differently formatted tables.

In most cases, reports are filled in based on a template. To accomplish this, the spreadsheet specifies a flexible area selection and parameter description system. The developer visually designs a template of the report’s appearance, identifying various areas in it and noting parameters that must be filled in with data. When the report is generated, the areas are combined with the data, thereby combining the report-building logic with the format. We should note that the report builder can either generate a report template itself or work with a template drawn by the developer. A formatting mechanism can be used that permits the user to actually use ready-made report formatting styles that can be applied to any report or develop his own.

It is very important that the spreadsheet is not simply a static tool for producing printed forms. It is a powerful interactive mechanism that can offer the user the ability to control multilevel groupings, perform drill-down, and automatically set column widths. The user can also save a report in various formats (html, xls, txt). In addition, a pivot table can be used in a spreadsheet for interactive analysis of multidimensional information; this enables the user to control the method of data grouping and the makeup of data displayed right in the report.

One of the spreadsheet’s most powerful capabilities is that it can also be a fully functional data entry mechanism. The spreadsheet can contain any 1C:Enterprise control, either in cells or above them, so it is used both as a report generating tool and as a data entry and editing tool. The use of these capabilities is especially effective when the very same form must be a report and also a data entry tool. For example, this could be a report on planning indicators with the capability for adjustment.

The data analysis mechanism, which enables the developer to include tools for mathematical analysis of management information in an application at minimal cost, is also very interesting. For presentation of analytical information, the developer is provided a set of various graph types, as well as a Gantt chart and a tree diagram. We will not detail these other report-building mechanisms in this article. However, we note that an important feature of 1C:Enterprise is that the developer is provided a full set of tools for generating reports of any complexity without the need to involve additional outside mechanisms. Naturally, the system does have a report wizard, which permits the user, by interactively selecting the required data and methods of presentation, to create a variety of reports, but it is important that the wizard merely assembles the report from available building blocks that can work together or separately. All the tools used are well integrated, operate in a single conceptual system, and make maximum use of each other’s capabilities. For example, a pivot table used in a spreadsheet using the report builder automatically modifies a query to the database as the user visually adds new levels of data grouping. This approach permits the development of fairly complex reports to be reduced to a few minutes.

Data Exchange and Distributed Infobases

One of the generally recognized “trump cards” of the 1C:Enterprise platform is its data exchange mechanisms. Originally, the decision to create a data exchange mechanism that was intended for geographically distributed information bases and did not require a continuous connection was prompted largely by Russian enterprises’ lack of access to quality high-speed telecommunications links. However, it became fairly obvious after a short time that these mechanisms corresponded to one of the most promising directions of development in world engineering thought in the area of corporate systems. At the moment, the application developer has at his disposal a powerful set of exchange mechanisms that can solve a whole range of problems. In addition to the traditional problem of supporting geographically distributed systems, this mechanism solves the problem of integration with applications, as well as the problem of creating complex heterogeneous information systems that include both the various applications on the 1C:Enterprise platform and other information systems.

The use of these technologies to solve the widest range of problems is made possible by the fact that the developer is not given a complete solution, but a set of mechanisms that can work either together or separately in any combination. These mechanisms include: working with XML documents, XML serialization of 1C:Enterprise data, messaging infrastructure, data modification recording service, exchange plans, and the distributed infobase mechanism.

A practically unique quality of this mechanism, in our opinion, is its high readiness for work. Setting up exchange in a distributed system involves practically no development costs. The standard capabilities permit interactive selection of the makeup of data that must be available for exchange and actually formatting and loading an exchange message. The system will automatically identify the need to transfer specific data, track the receipt of exchange messages and the need to retransmit data, resolve conflicts, and verify the integrity of uploaded data. However, the flexible capabilities for customizing the mechanism permit the setup of practically any exchange node topology with an arbitrary makeup of data participating in exchange and arbitrary conflict resolution rules. The exchange mechanisms minimize data transmission (only changed objects are sent) while also guaranteeing message resistance to loss, that is, they can operate both with and without guaranteed delivery.

The possibility that such an organic solution to the problems of integration and exchange in the platform is not due only to the exchange mechanisms themselves. This solution relies primarily on the common architecture of the 1C:Enterprise application. We have already said that the organic approach to distributed and integrated systems is provided by the object-oriented data manipulation technology used in 1C:Enterprise. The system of object used to manipulate data provides the standard capabilities of persistence of any data in XML format. In addition, the definition of standard application object prototypes enables the platform to automatically support a certain exchange strategy for each object corresponding to its purpose and standard use. The main problem in using most exchange and replication systems is that in their data description semantics, the system has no information about what packet size to use for exchange, how to resolve conflicts, or how to ensure the logical integrity and consistency of data. Accordingly, the developer must customize the exchange procedures in detail when using such mechanisms. The 1C:Enterprise exchange mechanisms operate at the level of standard application object prototypes, so they have all the required information from the start. The developer need only specify that a given type of object participates in exchange. Then the exchange mechanisms can perform all actions to record changes, generate messages, upload data, and resolve conflicts automatically. Of course, the developer can intervene and perform “fine tuning,” but this is required only in isolated cases.

Another important aspect of exchange mechanisms is compliance with future concepts of information systems integration. In developing these solutions, we paid special attention to the fact that today, only one developer of economic software can feel like “king of the mountain”

who has no need to interact closely with other systems. By “interact” here, we mean not only the ability to call some function from the other system or upload some data when necessary. We are speaking of building complex heterogeneous systems in which several dissimilar applications form a continuously operating team. Obviously, this direction will be one of the dominant trends in the development of economic systems in the foreseeable future. The 1C:Enterprise exchange mechanisms are very ready to use in the heterogeneous systems of today and tomorrow, not only because all exchange is done in XML. The most important thing is the ideological compliance of the solutions used with the concepts of corporate and distributed systems development. For example, these mechanisms can be integrated with specific solutions and also with technologies that exist today and are under development for controlling heterogeneous information systems.

The Application Updating System

Practically all software systems provide updating mechanisms to install new versions. However, for systems such as 1C:Enterprise, these tasks present a special difficulty incomparable with updating ordinary systems. The reasons are first, that economic systems are updated much more frequently than others, and second, that 1C:Enterprise permits the application to be modified on site. The latter feature presents special difficulty because whenever a developer releases an application update, it becomes necessary to synchronize the changes made by the developer with the changes made during deployment to account for the specific needs of the particular client.

To solve these problems, as well as all the other problems related to delivery and updating of applications, 1C:Enterprise implements a whole set of mechanisms to meet the needs of both the developer and the user. The functionality of these mechanisms encompasses the entire technological process of delivery and support: preparation of delivery and update release files, preparation of installation delivery kits, Internet publication of updates, automatic update search by users, execution of updates, control of the content of support at the configuration object level, etc.

One of the most essential parts of updating technology is the mechanism that synchronizes changes made during deployment with changes made by the application vendor. This mechanism provides a powerful régime of application change comparison and change synchronization control. The administrator or developer can customize the synchronization of updates right down to individual objects, individual properties, and individual module procedures. For example, he can mark objects that he plans to support himself in the future, and they will not be updated. If objects must be updated, he can customize the priority of aggregation in order to simplify the synchronization of changes.

The updating mechanisms permit construction of a complex multilevel support system. Vendors of more specialized vertical solutions can obtain support from developers of all-purpose solutions, and in turn provide support to users. The user can, for example, use an application certain parts of which are supported by different vendors.

It is important to note that the update mechanism is based on the fact that the application is described as a metadata structure, and standard application object prototypes are used. This permits the system to support an update strategy that meets the specific needs of installing the application, monitoring the logical integrity of the application, and providing the user clear information about changes.

As Well As...

Of course, the scope of this article does not permit us to describe all the interesting mechanisms of the 1C:Enterprise platform. Let us briefly mention several more points.

An application's entire user interface can be described in several languages. Each user can work with the system in one of the interfaces installed in the application. The exporting of all strings into separate objects is not used. All labels are edited in several languages "on site"—in forms, print templates, menus, etc. The developer can switch the current language and edit the form interface in another language, and can edit all options of a particular label in different languages. He is also provided a powerful interface translation mechanism that collects all the labels of the application into a common list and permits identical labels to be translated at once in all interface elements. The application's interface translation mechanism is part of an overall strategy of internationalization that also includes switchable system interfaces, storage of all strings in Unicode, date and number formatting in accordance with the peculiarities of the various countries and languages, and sorting rules that account for national standards.

An application developer is given the ability to transfer the execution of algorithms related to business logic to the 1C:Enterprise server. On the one hand, this enables him to manage the load distribution between the client and server, but on the other hand this ability is provided on fairly simple terms and requires no special knowledge of the building of three-level architectures, secondary protocols, etc. The system takes on all the technological details and provides efficient use of server resources by maintaining stateless models, caching, and dividing system resources. Another important feature of this mechanism is the ability to run certain procedures that affect confidential data in the name of a user who lacks broad rights of random access to the data.

The 1C:Enterprise access rights protection system implements the possibility of customizing access restrictions at the level of individual records, as well as table fields. This makes it possible, for example, to grant a user access to a list of employees only within his department, and salary and marital status data only for his immediate subordinates. The customization of such restrictions is done by setting an arbitrary restricting condition. The system will automatically monitor all user actions and the developer need not take these restrictions into account in various interface modes and business logic algorithms. Depending on the task being performed, the access rights control system can either completely ban access to data, if the user has requested records inaccessible to him, or simply exclude the inaccessible records from the selection, providing all accessible information.

One powerful developer's tool in 1C:Enterprise is the application comparison and aggregation mechanism. This mechanism permits comparison of both "related" and heterogeneous applications. It can be used either to analyze differences or to transfer part of the functionality of one solution to another. The mechanism provides a convenient visual representation of the differences and the flexible ability to customize the comparison logic. The ability to implement an efficient comparison mechanism relies primarily on the presence of a metadata structure and application object prototype system in 1C:Enterprise applications. For example, for related configurations, the comparison mechanism automatically compares even named objects, using the internal metadata identifier. On the other hand, the developer can manually customize the matching of objects independently of similarity or difference in their names, and the comparison mechanism will account for the compared objects when comparing all references to these objects in all the application's objects. In addition to the visual presentation of differences in the metadata structure and texts of templates, the comparison mechanism contains tools for visual presentation of differences in interface forms and print templates. Thus, by using the metadata structure and tools for visual comparison of interface objects, the 1C:Enterprise comparison mechanism can perform substantially more tasks than comparison mechanisms based on comparison of program source code files.

1C:Enterprise has a mode for recording user actions: the recording log. It is designed to store and analyze users' work histories. The system has several recording levels, which can be run automatically without further effort by the developer. Naturally, the system primarily records the

beginning and end of users' work sessions, administrative actions involving the infobase, and errors that occur in the course of the users' work. Because all changes in the database are made exclusively by object technology (through objects responsible for changing data), the system can automatically record data changes as well. This requires only selecting the appropriate level in the recording log setting. From then on, the system will record any data changes regardless of how they are made. In addition, the developer can implement the entry in the recording log of any other entries that the system cannot record automatically, for example, a record that a fax was sent, or that a report was generated. Based on the accumulated information, the administrator is provided various opportunities to view the record filtration. It is important to note that the recording log is implemented as a separate system mechanism. It does not use the database that contains the 1C:Enterprise infobase to store information. This enables a record to be kept in the recording log without creating substantial additional load on the system and without harming the simultaneity of users' work.

And So...

In our description of selected mechanisms, we tried to note that many qualitative capabilities of some mechanisms rely on the presence of key features of other mechanisms on the platform. For example, many mechanisms rely on the use of metadata and the presence of an application object prototype system. Another example is the characteristic mechanism and the user-expandable analytical sections of bookkeeping, which are based on the database structure's ability to describe fields of compound types. Thus, each of the 1C:Enterprise innovations has a certain intrinsic value, but at the same time it is most interesting to discuss them as an integrated model.

Once again, we can cite several general principles that form the basis of 1C:Enterprise and have been mentioned here as they apply to particular mechanisms:

- The platform presents finished answers to all the application developer's questions, from how to reflect subject-area data in the infobase to administrative issues, and even the creation of releases and the updating of applications. Accordingly, the developer need not master diverse technologies or address issues of their combined use.
- All development, from data structure to interface and support issues, is done in a single conceptual system. This substantially expedites the training of specialists and improves their work efficiency.
- The system's architecture and development tools are implemented so as to remove the developer as much as possible from technological details and allow him to concentrate on solving problems directly related to the subject area.

In our judgment, it is specifically the presence of a unified, thorough, high-level model and corresponding technologies that reduces the costs of application development and support by an order of magnitude.

When we began working in this area in 1996, and after we created the first version of the platform, we were not sure that this was the right way. Since 1996, our concept has developed successfully, and as of today it is embodied in the 1C:Enterprise 8.0 platform. Strictly speaking, the world still does not have many specialized technologies aimed at developing corporate and business applications. However, it is now becoming obvious that this direction is a strategic way of developing business software.

In the judgment of many specialists known to us, the situation that has emerged in Russia and the former Soviet Union with the presence of a business solution development and support industry based on a single technological platform is practically unique in the world. According to our observations, there is no analogous specialized platform on which different companies have created so many different mass business applications covering a substantial part of the market in

any region. Accordingly, our solutions are of special interest to many specialists because they demonstrate the real benefit of this approach.

In this article, we have attempted to list the most interesting and original solutions in the 1C:Enterprise platform. However, in doing so we have tried to show the reader that all solutions in the platform, although they may be of intrinsic scientific value, are primarily based on the use of a single model and a single architecture that permits the achievement of organic coexistence and maximum benefit from each mechanism.

Naturally, all the judgments offered here (including our analysis of contemporary trends) are merely our opinions and the results of our research. In no case do we pretend to absolute truth, which in these matters probably does not exist. Many readers will probably have their own opinions, different from ours, on the problems discussed here. We will be happy to receive critical comments on the article and other opinions.
