GLOBAL OBJECT INTEGRATION INFRASTRUCTURE SUPPLEMENTING SOA WITH DATA MANAGEMENT

Perspectives of Creation

Vladimir V. Ovchinnikov, Yuri V. Vakhromeev and Pavel A. Pyatih
Fusionsoft, May 9th St., Lipetsk, Russian Federation

Keywords: Object integration infrastructure, distributed objects, global object space.

Abstract: The paper considers a way of data-centric object integration supplementing SOA with data management. The key aim of the proposed approach is to provide unlimited scalability of object integration into one object space, having the classic typification mechanism, the general security and transaction management. Declarative publishing of objects in the space make it possible to read, refer and modify any of them in the general way, in secure and transactional manner in spite of where they are stored actually (RDBMS, XML, etc.).

1 INTRODUCTION

The current level of IT-technologies enables advanced solutions: complex systems intercommunicating one with another using open standards. IT industry has evolved a lot during the last one-two decades; however information itself still remains fragmented in the Net. There are mature tools for human-convenient information representation by means of a single program – WEB browser. But you understand that it has nothing in common with a consistent integration on the level of information/data; it's only accessing to systems through one window. Absence of consistent integration is apparent when a user, taking some data from one system, tries to look for related data in an adjacent system. This is impossible if only both vendors do not endow the systems with appropriate functionality initially.

If we would have said that IT society did nothing to overcome the problem, we would be wrong. Many things are being done:general system intercommunication standards and architectures (SOA, for instance) as well as particular integration standards for specific application domains are being developed. However, current efforts do not seem to be enough since standard development is complicated by two factors: considerable quantity of interested parties and high complexity of system integration technologies.

In this connection the question arises as to whether the technological component can be simplified by creating infrastructure for consistent system integration? Of course, you can say that the infrastructure already exists – SOA-based solutions. Indeed, SOA may be used for high-quality integration of a number of systems under one program interface, but efficient solution of the task is only possible on the basis of open standards for the appropriate application domain, which do not always exist. If there is no standard, each system vendor has to rely upon their own solutions. As a result, we acquire a set of separate systems, and intercommunication between every pair of them should be developed anew on both sides.

SOA gives facilities for system integration, but integration can not always be done in sufficiently simple and scalable manner. Real simplification of system integration process may be achieved only when systems being integrated start *working in one object space* with common mechanisms for accessing and referring to objects. Only this space may eliminate boundaries between systems by making the systems operate on common data with no need to develop separate modules for intercommunications of adjacent systems. The integrated object (data) space may simplify the system integration process significantly. In fact, an integrated meta-system is being created by that.

V. Ovchinnikov V., V. Vakhromeev Y. and A. Pyatih P. (2008).
GLOBAL OBJECT INTEGRATION INFRASTRUCTURE SUPPLEMENTING SOA WITH DATA MANAGEMENT - Perspectives of Creation.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 272-279
DOI: 10.5220/0001882802720279

Apparently, this data-oriented approach and SOA, being behavior-oriented, do not contradict but supplement each other. The integrated object space brings data-oriented system integration up to the new technological level since it hides all the data access specifics and gives the unified interface for object processing. The space can be extended in flexible and transparent manner without implementation of complex integration projects within existing systems. By the same reasons every SOA service benefits from the object space: services can use common data without any restrictions.

The approach of object space creation is not new. The very approach is used in solutions integrating several relational schemas into one. In this case a user interface developer can use the integrated schema to create seamless GUI and do other data processing in sufficiently simple manner. But these solutions have scalability restrictions bounded by a company or a holding company. It is impossible to extend the set of spanned systems endlessly; it will result in efficiency degradation for the whole integrated system sooner or later. But we tell here about the integrated object space with unrestricted scalability which is able to publish any quantity of objects and systems.

Of course the scale of tasks to be solved for creation of such global object space raises doubts in their principal feasibility. First, we should prove that the space may be created technically. Intuitively, it seems apparent that it is impossible. Second, organizational steps of such project are infinite. For such system to start operating in full, many vendors should be involved, which requires enormous efforts from the whole IT society.

We should note that further reasoning about the integrated object space is not only theoretical. Ideas discussed here underlie the product EntryService of Fusionsoft Company which is accessible here: http://www.fusionsoft-online.com/entryservice.php.

Let's consider the main doubts arising when we start talking about the integrated object space on a global scale. The first doubt is how to ensure sufficient scalability for such space if there are so many, inestimable, objects to publish? Whether the space preserves its efficiency during extension? The apprehension can only be dispelled in more detailed discussion below. Here we can give the short answer: hierarchical organization of objects and systems, like it is done in X.500/LDAP and DNS, gives all necessary means to solve the scalability problem.

The second doubt is what to do with those objects which are already stored in different data bases and used within existing systems? For the objects to be used in creation of new integrated systems, they should be logically mapped to the integrated object space without physical copying. In this case, actions over the objects done within the space should be immediately mapped to actions over the stored objects, ensuring information consistency. This approach lets create new systems which are initially integrated with other systems without a separate integration project.

The third doubt is whether it is possible to ensure the sufficient level of information security for such integrated object space? The short answer is yes since objects can be published at an information owner's place, and so both special security facilities as well as standard ones (VPN, etc.) can be employed.

And the main doubt is how can the object space be useful for resolving integration problems? If two systems were created separately, then the answer is not comforting: they can not communicate one with another without revising. Revision is necessary regardless of an integration technology used since data relations should be created between objects of different systems.

Then why we introduce some object space if existing systems can be only integrated using replication, ETL and other means as before? First, the approach being discussed is mainly oriented at development of new systems integrated initially by the fact of creation. Systems created newly and working with objects through the object space are seamlessly integrated with all systems worked with the same objects before. This is apparent from the fact that any system may read, modify and refer to objects regardless of how and where they are stored, and so end users can get any interesting information from any system without restrictions.

The above does not mean that the approach can not be effectively employed for integration of existing systems which was not built over the integrated object space initially. Besides data integration for some systems itself, new user interfaces are required to work with integrated data as a whole, only then end users can benefit from it. Of course, modules for existing applications, working with data of adjacent system directly, can be created, but this functionality should be added twice for every pair of systems being integrated, to applications of both systems. It is often so even if a user interface is built for WEB browsers.

This problem can be solved by creation of the integrated object space over the systems and by development of user interfaces and data processing

algorithms over the space. Doing so, we achieve integrity of user interfaces and make it simple to extend the cluster of integrated systems. When other systems are added into the cluster, their objects are to be published in the object space for end users to become able to work with them. If new objects are of the types being already shown in GUI, then no additional programming is necessary. Of course, if some types are not covered, then GUI should be revised once for every new object type added.

There is no need in the integrated object space when isolated systems are created. But when we concern integration of several systems into one meta-system, a consistent object space, based on one or another technology, becomes needed, especially when systems being integrated are created at different time. Having no such integrated object space, solutions will be scrappy to a greater or lesser extent.

There may be a lot of different approaches to organization of an object space. Some of them are implemented as high-quality products (for instance, approaches to relational schema integration). Further we will discuss our vision of this complex question and try to disclose motives for every architectural decision made, not claming to uniqueness of solution.

The key property of our approach is its unrestricted scalability which can be useful within one company as well as within the Net as a whole. Organizational problems concerning the global character of the object space should be considered separately and are not discovered here.

2 SCALABILITY OF THE INTEGRATED OBJECT SPACE

Unrestricted scalability of the integrated object space was declared above. How can we achieve that extension of the space will not result in its degradation in time? Such scalability is possible only in case of complete diversification of information flows between the object space and clients using it. It means that if all data flows go through one or bounded set of network nodes, then scalability of such solution will be limited by bandwidth to these nodes. The limit will be reached sooner or later. So we conclude that there should not be such nodes. But how this decision goes with object space connectedness? Does not it mean that the object space will become fragmented?

We use the following terminology in our solution. A part of the integrated object space placed at one host is an object service providing the program interface being general for all such services. Every service manages its own part of the object space and implements functions for accessing, modification and referring to objects published with it. Integrity of the object space goes from hierarchical service organization with a single root, which helps to find any service when necessary.

The service hierarchy has part-whole relationship semantics: a service may be terminal one working with stored objects directly or intermediate one combining some other services. Terminal services manages their objects exclusively, one object can not be managed by several terminal services. Intermediate services combine subordinate services just logically and do all work on them through subordinate services rather than directly. One subordinate service may be part of an only intermediate service.

So all services are parts of the root service directly or indirectly. Does it mean that we violate our requirement and the root service is the one participating in all information flows between clients and the object space? No, it does not. Existence of the root service does not forbid using the service containing an interesting object directly. Then does it mean that a client should know the service managing the interesting object before using it? It's not true as well.

The program interface of all services has two parts: inter-service and client. The inter-service part backs service hierarchy and inter-service communication, while the client part does not concern inter-service organization mechanisms at all. From the client's point of view every service gives means to use any object of the object space with no any restriction: it is sufficient to connect to any service to *get access to any object published*. How can we achieve both accessibility of all objects and absence of a single node transmitting all information flows through?

Accessibility of objects through any service is backed by inter-service communication when an unknown object requested by a client is to be found. After an object is found, it is accessed through the terminal service where the object is stored, which gives significant diversification of access ways to objects with known placement. Therefore the object search algorithm gets critical importance: scalability of the whole object space depends on diversification of information flows in the search process.

One of solutions could be creation of object registries for every service. Terminal services could have the registry of objects stored in it directly, and intermediate services could have the registry of objects stored in nested services. But this solution does not provide the necessary level of diversification for access ways, and here is the reason.

On the one hand, such registries could ensure for an object to be found in sufficiently efficient manner since we could go up the service hierarchy to the service where the placement of the object is registered (like searching an address for a domain in DNS). As a result, the root service would be used only when objects to be found and the client are in "opposite" sides of the object space.

On the other hand, this approach does not give the necessary level of information flow diversification. If the "opposite" part of the object space contain a considerable amount of objects looked for, then density of data flows to the root service and its nearby services will rise significantly. As a result, when clients show noticeable activity, then the object space will degrade inevitably. Moreover, objects are not as permanent as associations between domains and addresses. Objects can be created, deleted, or modified actively. And having taken into account that this approach requires to propagate object registries from terminal services toward the root one, we conclude that this propagation itself will result in absolute efficiency degradation for the object space regardless of any other aspects.

Therefore this approach, based on object registries, can not be used in the integrated object space being global. But is there any alternative which does not result in the object space's degradation during extension? Yes, there is.

The key to alternative solution is such way of object identification within the integrated object space which ensures efficiency of looking for the terminal service storing the object. Of course, the simplest decision here is to use a terminal service's address as a part of object identification. But this decision will impose strict restrictions on object space flexibility. For instance, if one decides to split one service on two or to move some objects between services, then it will require modification of all references to the objects being moved, which is impossible in general case since the amount of references may be enormous and their placement is not always known. Therefore we can not use the terminal service as a part of object identification.

But how else can we structure object identifiers to find their terminal services efficiently? The answer is usage of hierarchical identification where all object identifiers form a tree from one root, and every node of the tree knows the service managing objects identified by this node and nodes nested to it directly or indirectly.

How efficient can be search in case of hierarchical identifier, will it degrade like the version with object registries? First, the search is done once for every identification tree node and not for every object covered by the node, which reduces information flows significantly. And second, nodes rarely change association to the services backing them. It enables local caching of this information within any service placed on the way between a client and the root service. Therefore we can conclude that services can be found using the similar algorithm as looking for domain in DNS. So service search efficiency is comparable with domain search efficiency for DNS.

So the hierarchical identification solution gives diversification of information flows during search for a service as well as during work with final objects . The level of diversification is sufficient for the integrated object space to function efficiently with unrestricted scalability.

And now the question arises: how to implement the identifier hierarchy in the most rational way, so that efforts for its creation could be minimized and identifiers could remain stable when moving objects? When answering the question, we use one apparent observation: objects and object properties form the part-whole hierarchy. For instance, an object of the type "person" can have the property "passport" which in its turn is an object and can have its own property "passport number". So any object can be considered as a node being parent for all its nested properties, and each property, in such interpretation, is an object, and so it can be a parent node for other properties. And so forth.

If we build such hierarchy from one root object, then hierarchical identifier of an arbitrary object may be formed as the path from the root object to the given one. How should we identify each step of the path in that case? If we talk about objects and their properties, then it is natural to use property name since it is well-known that properties have unique names within objects, and so they identify each branch of the tree unambiguously. Therefore, the unique identifier for the object "passport number" may be /People/Vladimir/Passport/Number, where "People" is the property of the root object,

"Vladimir" is the property of the object "People", and so on.

Such approach let us reduce the identifier hierarchy to the object hierarchy and to avoid artificiality of the identifier hierarchy. Identifiers acquire substantial meaning and may be used for referring by systems as well as by people. Efforts for creation of such hierarchy are equal to efforts for creation of the object structure, no more.

So, taking into account these additions concerning identifier organization, we can ascertain that the whole object space, being the object hierarchy now, can be split on several sub-trees having its root object associated with some service used to store this root and nested objects. Sub-trees can be extracted in arbitrary way with the single restriction: every object should be associated with one of services directly or through its ancestors. Therefore, it is possible the situation when sub-trees form another tree, where some branches give rise to other sub-trees for other services. As a result, the same service may be intermediate, having some subordinate services, as well as terminal, storing some objects directly. It gives significant flexibility of distribution of objects among services, and so it gives load-balancing flexibility and high scalability of the object space as a whole.

Sub-trees may be joined and split on parts without changes in object identification. As a result, identifiers remain stable when moving objects. It is sufficient to assign a new service to some higher object to get access to it and all subordinate objects through this service, without changes of identifiers. By that we acquire higher flexibility of object distribution among services.

Also it is important consequence that the object hierarchy can be used to solve research tasks and not only getting of specific objects by known identifiers. Thus a user can browse the object tree investigating what objects exist and what properties they have. Moreover, the hierarchy can be overbuilt with a query language like X-Path/X-Query. Efficient execution of queries of such language is a separate question, and it requires of course corrections in light of global character of the object space.

3 OBJECT TYPIFICATION

The object space can have no typification theoretically. But how to understand that two objects describe the same entity, for instance, people, in this case? Since every object has properties, we can try to extract an entity from the fact of presence of some typical properties, for example, for the entity "Person", the properties could be "Name" and "Age". But stars have names and ages too, which does not mean that stars are people. Therefore properties are not reliable for this.

There is only method to resolve the problem of associating objects to entities: we should introduce object typification. Every object in this case should be related to some type which should unambiguously determine the appropriate entity of the real world. For instance, objects of the type "Person" are not starts, but are people, and vice versa. Object typification defines the way of interpretation for all objects of some type. For example, the task of analysis of social bounds has sense for people, and the task of distance analysis has sense for stars, but not vice versa.

So, to understand entities related to objects, we should define a type for every object existing in our space. But it is not enough. Correct interpretation of an object means correct interpretation of its properties, which requires that the same property should be equally named for all objects of the same type. Therefore, every type description should contain the structure of typical object of the type. Only in this case all objects of one type may be processed in common way, reasoning from commonness of their structure.

How should we store object types in the object space to access them efficiently, not less efficient than for objects themselves? The solution is simple: we should represent object types as a special kind of objects. Object types, being objects simultaneously, relate to the special type corresponding to the entity "Type". Such recursive type definition let process them as any other object.

What if an entity is a particular case of some other entity, for instance, the entity "Sportsman" is a particular case of the entity "Person": what type should be defined for the particular entity in this case? The problem is that we can not relate one object to two types simultaneously, but any object of the type "Sportsman" should be related to the type "Person", what can we do? To resolve the contradiction the concept of type inheritance should be introduced.

Inheritance of some derived type from some base type means that every object of the derived type is also an object of the base type by definition, and so it has all the properties of the base type. In our example, we should introduce inheritance of the type "Sportsman" from the type "Person". When inheriting, there is no necessity to repeat the structure of the base type within its derived types:

inheritance of properties is implied. Therefore, in spite of the fact that the type "Sportsman" has only additional properties, like a sport kind and achievements, every object of the type "Sportsman" also has all properties of the type "Person", like a name and age.

So to find out is an object related to some type, it is insufficient to get the object's own type: all ascendants of the object's own type should be analyzed too. If the ascendant type set contains the requested type or the requested type is equal to the object's own type, then the object is considered to be related to this type regardless of its own type can be not the same. In our example, according to this rule, every object of the type "Sportsman" is an object of the type "Person" inevitably.

May the object space contain several types for the same entity? Yes, it is possible if types are developed by different companies with no coordination. In this case, objects being of the same entity but of different types will be interpreted differently. The object space, being integrated physically, starts splitting on parts unconnected one with another logically.

There are two solutions for the problem: organizational and technical. The organizational solution includes preliminary standardization of types by some standardization institution. As a result, different companies use the same set of types initially. This approach saves resources for applied system development dramatically, but it is not always possible since standard development requires noticeable amount of time and coordination among many interested parties. So, standards are created, as a rule, after several alternative solutions enter the market.

The technical solution may be applied even after system creation, but it is much more complicated. According to this approach, a standard or a private treaty among companies is created on the basis of existing scattered types. The standard defines a new set of standard types covering the appropriate application domain in full. This organizational step is inevitable if we wish to get complete system integration.

Further, every system using its own types should inherit the types from standard ones; a module backing the inheritance should be added. This task does not require modifying a system itself, it can be solved on the level of the service integrating the system to the object space by providing access to objects of the system. As a result, every special system uses its own types to manage its objects, but it is also possible to use the objects out of the system with standard types. It allows us decouple next integration steps for different systems.

The third step is the most complicated one: a new version, based on standard types only, is to be created for every system. Of course, there is an alternative: to convert external objects of standard types to internal types of the system (using ETL or "on the fly"). But this solution has limited scalability, timeliness, and efficiency since the quantity of objects to convert may grow uncontrollably. Efficiency of the system may degrade completely in time, and administration of the converting process may become overly expensive. This approach, of course, can be used in some special cases, when, for instance, the system is not expected to evolve, but it's more long-sighted and scalable to issue a new version of the system.

At the end of the third step, after we have used standard types, we should find objects which instances were multiplied in different systems, reduce the instances to one for every object by removing surplus ones, and replace references to surplus instances by references to chosen master instances. As a result, we restore integrity of the object space: every object is represented with one instance of a standard type, and objects are interpreted in the same way in all systems.

Here the question arises: could such integration result in reducing efficiency of existing systems because we start using remote objects? On the one hand, of course, yes: integration requires some additional resources, it is inevitable. On the other hand, efficiency problems may be resolved after, for instance, a local service caching often used objects may be installed. As a result, somewhat reduction of remote object access efficiency becomes apparent only in case of active modifications of the objects by the remote side.

It is evident that the approach described above is allied to approaches of object-oriented programming, conceptual schema design, relational schema integration, ontology design and fusion, but we can not affirm that it is one of them. The approach absorbed all that is necessary for qualitative system integration within one integrated object space.

4 SECURITY OF THE OBJECT SPACE

The integrated object space may be used for solving critical tasks only if it gives flexible and reliable

technology for ensuring object access security. First of all, it requires permission management for reading, referring and modifying objects.

There are three levels of permission management: object space, intersystem and network. The object space's security management enables for authorized people to do certain activity over some objects. When necessary, permissions can be logically propagated down the object tree to simplify administration. It lets us describe the most general permissions once on higher levels of the object tree (as a rule, permissions of reading and referring to public objects), and special permissions on lower levels for nested objects.

Before permissions can be verified, a service should know the user, on behalf of which some activity is being carried out. For that, the user should connect to the object space and go through the process of authentication. But where to store user information, taking into account that storing user names and passwords at one place is not convenient, scalable, or reliable solution? In this case, by increasing the quantity of users, their names will tend to be duplicated. Moreover, access to user information will be done though a single node. Loosing connection with this node will result in unavailability of the object space for all users in the whole. So, the solution with a single node is not appropriate at all.

In our solution, users of the global object space are described as objects of the special type, and the objects can be stored at any part of the space. Of course, the solution does no exclude usage of one or several services for centralized user management. But at the same time, any company, which decides to manage user accounts by any reason, can do that within its own services. One of motives of such decision may be ensuring accessibility of the object space for some users, for instance, for employees of the company.

It is evident that user passwords should be encrypted by means of one of known digest algorithms proved their effectiveness. In our solution, for instance, a user can choose the algorithm in case he (she) wants to choose.

To simplify administration, the object space should let group users into roles: groups of users solving similar tasks. It allows permission management on the level of roles, not separate users, which increases flexibility of the administration process. So, for an administrator to give some standard permission set to a user, it is sufficient to give the role having the permissions.

The second level of permission management is the systems, having services operating over them, themselves. The systems may have their own mechanisms of authentication and authorization, as a rule, in case of DBMS usage. The object space should give the possibility to use the mechanisms as another security layer.

At this level, a name and a password to access a certain external system may be assigned for a role or a user. To make administration simpler, the assignment can be done automatically with a special tool. Then an administrator can manage permissions for the role/user using system-dependent tools on the level of the system itself, for instance, DBMS. If a permission set should be given to any user connected to a certain external system, then a guest account should be provided for the system, which is to be used for unregistered users.

And the last level of security is the network level. Let some company consider necessary to create a system as a part of the integrated object space, so that it can refer and use external objects published globally, but intersystem objects are to be accessible within local network (or VPN) only, for example, because of financial character of the system. In this case, access to the local service can be restricted not only on the level of the object space or DBMS, but also on the network level by forbidding any external connections to the object space service. As a result, the local service will be operating as a part of the global object space, but it will not be accessible from the outside.

5 TRANSACTIONS IN THE OBJECT SPACE

The object space can be considered as completely integrated only if it provides a single transaction management mechanism. A user (or a client program working on behalf of the user) should have possibility to make a sequence of changes in one transaction regardless of whether objects are within one or several services. Neither user nor client developer should be concerned about whether a transaction is distributed or local: it is to be ascertained by actual distribution of objects being changed.

Therefore, all services of the object space use a family of transaction managers comparable each with other, having common rules of intercommunication, including the rules of transaction identification, two-phase distributed

transaction commitment, etc. By initiating a sequence of object changes using some service, the client activates the transaction manager of the service. If objects of different services were affected, transaction managers of the services get, transparently, the context of the distributed transaction the changes are within, and manage the local changes as a part of the distributed transaction.

Distributed transaction management is to be done according to the model X/Open DTP. The model says that if a system storing objects has its own transaction manager, for instance, DBMS, then transaction control over the objects may be delegated to the manager. In this case, the appropriate service's transaction manager may do no work itself, but delegate it to the transaction manager of the subordinate system.

Questions of reliability and transaction durability may be solved on the level of a service's transaction manager if it does full management, journaling etc., or else on the level of the subordinate system's transaction manager.

6 SUMMARY

We hope that we proved technical practicability of creation of the global object space. Ideas stated here were checked in practice and were used in the product EntryService of the Fusionsoft company, which you can get here: http://www.fusionsoft-online.com/entryservice.php. We are using the product for creation of the object space now. We welcome joining us in this task: info@fusionsoft-online.com.