ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

B Novikov, University of St Petersburg, Russia and J W Schmidt, University of Hamburg, Germany. (Eds)

Advances in Databases and Information Systems, Moscow 1996

Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS '96). Moscow, 10-13 September 1996

A Distributed Architecture for Integrated Access to Legacy Database Systems

Silvio Salza, Antonio Massari and Massimiliano Bruno

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

A Distributed Architecture for Integrated Access to Legacy Database Systems

Silvio Salza, Antonio Massari and Massimiliano Bruno
Università di Roma “La Sapienza”
Roma, Italy

Abstract

Workflow integration is a problem common to many organizations, that need to develop and run applications composed by multiple related tasks, possibly heterogeneous and performed on or by heterogeneous processing entities. Often some of the systems to be integrated have a very large autonomy, and concede only very limited visibility of their internal organization. The limit case is given by legacy applications, i.e. old, typically mainframe based applications designed with little or no interface with other applications. In this paper we discuss the design and the implementation of a prototype system for development and the management of such distributed workflow applications. To carry out the integration process we propose both a graphical formalism to represent the access procedure to legacy host, and global application that involve the access of several host. This formalism allows to represent such concurrent processing application in a totally structured way, and includes provisions to represent different kind of parallel processing, and the handling of asynchronous exceptions.

1 Introduction.

Many organizations make use of different, independently-developed information systems to automate their business processes. Certain multi-task activities require the participation of different systems and databases, where the component tasks are applications accessing data from large and autonomous information systems. A typical case is when such applications run on a *legacy system*, i.e., an old, main-frame based system characterized by large autonomy and designed with little or no interface with other applications.

A *workflow* in an enterprise typically involves performing multiple, related tasks, which can be heterogeneous and performed on or by heterogeneous processing entities. A workflow can be simply defined as a set of tasks (also called activities or steps) that cooperate to implement a business process. A good workflow technology can also provide a way to make a good use of past investments by allowing integration of legacy systems, and the flexibility to support significant organizational changes and technology evolutions. The specification and coordination of a *heterogeneous workflow* (intended as a multitask activity carried out on heterogeneous systems) in a scenario where the autonomy of the participating systems is a primary constraint, requires to deal with the following issues:

- *parametricity* with respect to the different interfaces through which the tasks are submitted to the processing entities;
- *data exchange* between tasks which may involve different formats for different tasks;
- *exception handling* especially for systems whose behavior may be not completely modeled or known.

A number of different approaches have been followed in the literature [1],[4],[6]. The METEOR system [8] supports the definition of individual tasks, both transactional and nonelectronic operations, including exception handling. In the work [3] a conceptual model for describing workflows specifically focused for interaction with external sources of information is presented. In [7], a language is proposed for describing (possibly nested) multi transaction activities, and for specifying the flow of data between different modules. The ATM approach ([4],[5]) includes an extended nested transaction model and language for describing long running activities. Such a description includes a procedural

Figure 1: A Local Host Query Access Schema

static specification of the high-level workflow, and rules (triggers) for the "dynamic" evolution of the workflow. In the ConTract model [11], a long running activity is modeled as a combination of *script* and *steps*, where scripts deal with the conceptual workflow and steps deal with individual tasks.

Most of the various approaches focus on the aspects concerning the models used to express the semantic of the workflow and they do not consider particularly restrictive hypothesis regarding the autonomy of the information sources and the visibility of the access methods.

In this paper we discuss the design and implementation of a prototype system whose main objective is to allow the specification and the coordination of a distributed workflow mainly involving the execution of applications running on old mainframe based legacy systems. In our approach the visibility of the data sources is limited to the end-user interface normally presented for a character-terminal access, thus pushing to the limits the assumption of autonomy made for the external systems. The prototype system was developed within a research project for the automation of complex procedures in some departments of the Italian public administration. The system is currently under testing on a sample application (complex legacy database search for police investigations). The prototype is written in C++ and is based on a CORBA platform.

The paper is organized as follows. In the next section we describe the typical interaction scenario between a remote user and a legacy application. Then in Section 3 we give a brief overview on the distributed architecture that we propose to support the integration process, and in section 4 we introduce the notions of *local* and *global workflow*. The prototype implementation of the distributed architecture is presented in Section 5, and the main characteristics of the implementation language are discussed in Section 6. Finally conclusions are drawn in Section 7.

2 Remote user access to legacy systems

Our approach was triggered by a typical scenario, common to many organizations that need to integrate the access to information from different and heterogeneous sources, yet all characterized by similar restrictions in terms of visibility to the internal data structure and autonomy. Motivated by the growing relevance of such applications, we have defined a

system whose main objective is to allow the specification and the execution of a distributed workflow mainly involving the execution of activities running on old main-frames (legacy systems) [2].

The main characteristics of such legacy applications can be summarized as follows:

- they allow only predefined transaction, in particular read-only transactions, and we shall use the term of *local host queries* for them;
- they were designed to be accessed through a predefined schema consisting of sequences of masks for a character-based terminal.
- the user of such predefined queries is supposed to provide inputs for the execution as well as indications for determining the evolution of the application. Sometimes it may happen that the evolution is non predictable; i.e., the application decides which is the mask to present next.
- the response time of such applications may vary considerably depending on several factors, such as: speed of the connection, traffic on the network, load of the host etc.

In Figure 1 an example of such a predefined access schema is displayed. The example refers to a host which manages information regarding real estate owners; in particular, by means of a predefined local host query, the user can retrieve the list of the properties of a given owner, by specifying his or her Social Security Number (SSN) and, optionally, the zip code of the area where to restrict the search. The picture represents, at the end-user level, all the details of the interaction process, by showing the sequence of terminal screens and the flow of the control.

The access schema works in the following way: at the beginning the user is prompted for the SSN of the person he wants to inquire about; the host replies with a mask displaying the owner name and the total number of properties. If the selected person is not owner of any property the query just terminates; otherwise, if at least one item exists, the user may choose to display either the full list of properties, or only those located in a specific area that he should specify by supplying the Zip code. In this short description, because of lack of space some details of the interaction are omitted, such as the handling of timeouts, and exceptions in general. These are anyway crucial aspects in most application, and are in fact carefully contemplated by our integration methodology.

Many organizations and enterprises actually need to access information from different sources which is available in a form similar to the above example. Moreover, in general, we cannot assume to be allowed of accessing the hosts at a level different from the end-user interface designed for a character-based terminal. As a consequence, any agent (either a human or application) which needs to integrate the information from the various sources must necessarily deal with such cumbersome low-level details of the access schemas.

The main goal of the integration is instead to make it easy the specification of a workflow which involves the execution of legacy applications; workflow specification must take into account only the relevant aspects of the interaction with the legacy host, such as input and output of parameters, conditional flow of control, and the handling of the exceptions which can occur during the execution. The all other internal details of the application must be hidden.

3 The reference architecture

An high level view of the architecture of our prototype system is sketched in Figure 2. There are four main components in the architecture:

- the *host systems* where all the information resides, and that make it accessible only through legacy application, with access schemes of the kind of that discussed in the previous section;
- the *host servers*, one for each remote host, which hide all the non interesting details of the access mechanism, by eventually including also a layer of terminal emulation, and provide a homogeneous higher level interface, independently from the legacy applications;
- the *application clients*, which support the execution of *integrated applications*, i.e. with integrated access to data managed by several hosts;

Figure 2: The Reference Architecture

- a *communication framework* interconnecting all the elements of the architecture, and supporting a client server interaction schema; as we shall discuss in a later section this is actually in our case an object-oriented framework.

The host servers are the only part of the integration architecture which depends on the characteristics of the legacy application. Actually there is a further organization inside the host server, so that the heterogeneity is limited to *special host interface* modules, and the rest of the structure is standard. From the point of view of the application clients, the external applications are presented by the host server as a set of services with a well defined input-output interface.

Typically a host server provides two kinds of services:

- specific *data access services*, which depend from the external application, but complying with an uniform definition standard;
- *general services*, which are host independent, such connection, monitoring, etc.

The set of services provided by the host servers may actually be considered as a *Application Programming Interface (API)*, directly utilized by the clients for the specification and execution of global applications.

The internal structure of the host server allows to manage multiple client connections, and concurrent execution of multiple services. An optimization is performed and host interactions are queued on a shared set of remote host sessions, thus avoiding to start a new session for every client connection.

Besides providing services to the clients, the host servers also acts as active counterpart toward them, by sending to them asynchronous events when exceptions occur.

The clients support the execution of global applications, which are specified in a special Workflow Integration Language (WIL), that will be introduced in the next sections. Moreover the clients provide standard facilities for implementing the user interface. The implementation of both graphical and textual interfaces is supported.

4 Local and global workflow

To set up a common ground for workflow integration, we introduce an uniform formalism to describe the user-host interaction process, at an abstraction level that makes it totally independent from the peculiarities of each host and from the low level details, as terminal protocols, I/O formats etc.

With a graphic formalism, the interaction process between the end-user and a local host when submitting a pre-defined query can be described by a *Host Interaction Schema (HIS)*. A HIS is a directed flowchart, with two kind of

nodes that represent host *interactions* and the *user interactions* and edges representing the flow of control that, as we discussed in the previous section, may evolve either because of the host or the user initiative. Although very effective in explaining the idea, the graphical formalism is actually not too handy in nontrivial cases, we therefore have defined also a programming language to describe HISs, that will be discussed in a later section.

Figure 3: Sample Host Interaction Schema (HIS)

A sample HIS for the local host query presented in the previous section is shown in Figure 3. Square nodes represent host interactions and specify the input and output parameters, provided respectively by the end-user and the host at the beginning and the end of the interaction. A rounded edge node represents instead a user interaction, i.e. the human reaction to an host output that leads to the selection of a new host interaction, and to provide a set of input parameters to it. Both kind of nodes may have multiple outgoing edges whenever the host or the user may have several choices in selecting next interaction. For instance user action *U2* represents the user choice between asking for the display of all properties, or selected properties in a given area only.

The main purpose of the HIS is to allow easy development of new applications based on local host queries that substantially hide the details of the legacy application, and replace the user interactions with application language procedures. These may in turn eventually include a more sophisticated user interface.

The HIS is actually part of a more general formalism that allows to represent global applications requiring integrated access to several hosts. That means representing a concurrent processing application that includes as component tasks local host queries. For this purpose we extend the graphic formalism of the HIS to express a *Global Workflow Schema (GWS)*. A GWS is a structured flowchart which connects several component HISs by means of a set of *global control structures* that allow to specify the *outer* level of the flow control, i.e. the one connecting the local host interactions which form the basic blocks.

More precisely we may define a GWS as follows:

- a GWS is a *basic block* or several basic blocks connected by *global control structures*;
- a basic block is an application language procedure, or a HIS where all user interaction nodes have been replaced by application language procedures;
- an application procedure may include a GWS as part of it.

Figure 4: Sample Global Workflow Schema (GWE)

Figure 5: Global Control Structures

Note that the last point makes the definition recursive and allows the nesting of GWS. The five global control structures are shown in in Figure 5:

- the *sequence* and *condition* structures represent sequencing and conditional selection of blocks;
- the *fork* structure represents concurrent processing of several blocks;
- the *iterblock* structure represents iterative processing of the same block on a sequence of input parameter sets, which are given as an input to the structure itself;
- the *pipe* structure represents pipelining of several blocks; the input to the structure is a sequence of parameter sets, and each block in the pipeline produces in turn an output list which feeds the next one.

The last three structures are especially interesting because they allow to speed up processing of global application that needs to submit several local host queries and/or to repeat a query for several input values. Note that in the case of pipelining the processing of each component block does not need to wait for the full input list in order to begin. For both *iterblock* and the component blocks of a *pipe* a *concurrent* option is available to specify that more than one instance of the block can be activated concurrently, as long as input parameters are available.

A GWS is therefore a fully structured flowchart, since all global control structures have a single input a a single output, therefore is possible to define in the blocks local variables with a clear definition scope, that are consequently accessed by the application language procedures.

To better understand the idea, consider the sample GWS shown in Figure 4. The example is based on the HIS shown in Figure 3 and a second local host query *MRTG*, that, given the code of a real estate property, checks if there is a mortgage on this property. The global application prompts the user for a SSN, then activate an instance of the query *PRPTY* to get a full or partial list of properties. Then the query *MRTG* is used to perform the mortgage search on each element of the list. The icon on the upper left corner of the iterblock indicate that the search should be performed concurrently by activating several instances of *MRTG*, as long as the local host supports them.

A further interesting feature in the GWS language (only outlined here due to lack of space) are *triggers* defined in a HISs in connection with special events that allow to specify in the GWS how the global application must proceed if the event happens. These are especially useful to handle host connection failures, local host query aborts, and in general any kind of exception.

5 Prototype Implementation

As already discussed in Section 3 , our prototype system was developed as a fully distributed application, with servers modules managing the connection with the legacy host systems, and clients interfacing the end-users. A more detailed view of the architecture is shown in Figure 6, where the main modules are depicted:

- *clients* provide the end-user interface and manage the execution of global applications, as specified by the GWS;
- *host interfaces* manage the connection with hosts, and hide all the low level details, thus allowing a host-independent structure for the rest of the host server;
- *host monitors* are in full charge of the connection with the hosts, and give the clients grants to submit local queries;
- *transaction managers* manage the execution of instances of local queries, according their HIS.

Each host monitor keeps open several host sessions which are utilized to submit local host queries on the client's request. When a client during the execution of a global application needs to start a local query on a given host, it makes a request to the corresponding host monitor. This one, as soon as one of the open sessions is available, starts a transaction manager which will manage the given local query and directly communicate with the client originating the request.

Figure 6: The Distributed Architecture

The design of the distributed architecture was performed according to the *Object Management Group (OMG) CORBA 2* standard, and the prototype was implemented on a network of Sun and HP workstations running the Orbeline system by Post Modern Computing Technologies. All modules are therefore objects and the communication between clients and servers is carried out through methods. For example the client calls a specific method of the transaction manager interface for every host interaction in the GWS. The method allows passing of parameters and status codes in both directions. Furthermore the transaction manager and the host monitor may invoke methods on the client when special events arise, like triggers, timeouts, host query aborts etc..

According to the CORBA standard all interfaces are defined in the high level *Interface Definition Language (IDL)*, and produce through IDL compilation the *Stubs* and the *Skeletons* which are respectively the client and server part of the interface (actually libraries of functions to be linked with the client and server code).

6 The workflow integration language

Clients application are written in a special language called *Workflow Integration Language (WIL)*, with control structures that correspond to those introduced for GWS in Section 4 and provisions to declare local and global variables. The WIL is actually a superset of C++ and IDL, and is precompiled to generate C++ code. The whole process of implementing an application is sketched in Figure 7. Note that the host monitor and transaction manager code is substantially application independent and specializes only by linking to the skeleton libraries produced from the IDL definitions. In turn the WIL precompiler process both the WIL code and the IDL interface definitions to produce C++ code that is linked to the stub libraries. Producing a new client application based on already existing host servers, would require of course only writing, precompiling and compiling the WIL code.

A sample WIL program for the global workflow schema discussed in Section 4 is presented in Figures 8 and 9. More precisely Figure 8 shows the IDL interfaces for the host interaction schemas `prpty` on `host1` and `mrtg` on

Figure 7: Application Development

```

interface prpty {
  struct property
  {string prop_cod; string prop_type ;//...};
  typedef sequence <property> prpt_list;

  H1(IN string ssn, OUT long prop_num);
  H2(OUT property_list full_list) ;
  H3(IN long zipcode, OUT prpt_list partial_list);

  ORDER# [H1:START];[H2:H1];[H3:H1];
        [STOP:H1,H2,H3];#
};

interface mrtg {
  struct property
  {string prop_cod; string prop_type};
  struct mortgage
  {string prop_code;string mort_cod;
   string mort_type};
  typedef sequence <mortgages> mrtg_list;

  H1(IN property prop, OUT mrtg_list mort);

  ORDER# [H1: START]; [STOP: H1] #
};
    
```

Figure 8: The IDL interface

```

STARTG RealEstate
  DEFAULT
    { TIMEOUT_ATT: 5; TIMEOUT_TER: 30;
      TIMEOUT_LOC_TER:10}
  GVAR
    { BOOLEAN full,print_mtg;
      host1#prpty#prpt_list prop_list}
  START P: host1#prpty
    Ask_ssn(P#ssn);
    P#H1(IN P#ssn, OUT P#prop_num);
    IF (P#prop_num==0) SET(GlobalAbort);
  ELSE{ Ask_For_FullList(P#prop_num, full);
    IF (full) { P#H2(OUT P#full_list);
      prop_list = P#full_list; }
    ELSE { Ask_zipcode(P#zip);
      P#H3(IN P#zipcode, OUT P#partial_list);
      prop_list = P#partial_list; }
    }
  STOP P;
  Ask_For_Mortgages(print_mtg);
  IF (!print_mtg) DISPLAY(prop_list);
  ELSE { ITER_BLOCK (CUNCURRENT)
INPUT: prop_list
OUTPUT:final_list QUEUE_OF host2#mrtg#mortg_list;
  { START M[i] : host2#mrtg
    M[i]#H1 (IN NEXT(prop_list), OUT M[i]#mort);
      APPEND(M[i]#mort);

  STOP id2[i];}
  DISPLAY(final_list);}
STOPG;

```

Figure 9: Sample WIL program

host2. According to the OMG model, each interface represents the description of an object, and includes the definition of the data types that are used in the methods and the definition of the methods themselves. Each method in the IDL interface corresponds to an atomic interaction in the HIS, and explicitly defines the input and output parameters. The last section in the definition is not standard and defines the partial ordering between the HIS interactions, so that the WIL precompiler can check that in a GWS methods are invoked in the right sequence.

Figure 9 shows the WIL code for the GWS RealEstate of Figure GWS. The first section contains the declarations of global variables, and the default definitions for the timeout parameters. The latter can later be overridden by further declarations at the start of the HIS. The START command activates an HIS, i.e. an object of the class defined by the corresponding IDL interface, and associates to it an identifier that is thereon used to denote its methods and parameters. The rest is standard C++ code, plus invocations of methods of the active objects.

In addition to the standard C++ control structures, the global control structures introduced in Section 4 are also used. For instance the ITERBLOCK structure is used in the example to activate a HIS of the class host2#mrtg for each element of the input list.

7 Conclusions

In this paper we have discussed the design and the implementation of a prototype system which allows to perform workflow integration in an environment where the basic tasks involve the access to legacy applications, i.e. old, typically mainframe based applications characterized by large autonomy and designed with little or no interface with other applications. Such a scenario is typical in many organizations that need to integrate the access to information from different and heterogeneous sources, yet all characterized by similar restrictions in terms of visibility to the internal data structure and autonomy. Actually our prototype system has been designed within a research project for the automation of complex procedures in some departments of the Italian public administration.

To carry out the integration process we propose both a graphical formalism to represent the access procedure to legacy host, and global application that involve the access of several host. This formalism allows to represent such concurrent processing application in a totally structured way, and includes provisions to represent different kind of parallel processing, and the handling of asynchronous exceptions.

The system has a fully distributed architecture with server modules that interface the legacy applications, and client modules that provide the end-user interface and manage the execution of global applications. The prototype was designed implemented according to the *Object Management Group (OMG) CORBA 2* standard. The paper also discusses the main aspects of the *Workflow Integration Language (WIL)*, as superset of C++ and IDL, that we have specifically designed to write workflow integration application.

References

- [1] P. Attie, M. Singh, A. Shet, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings 19th VLDB Conf.*, Dublin, Ireland, 1993.
- [2] M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems: gateways, interfaces and the incremental approach*. Morgan Kaufmann Publishers, Inc., San Francisco CA, 1995.
- [3] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In *Proceedings of OO-ER conference*, Gold Coast, Australia, 1995.
- [4] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *Proceedings ACM SIGMOD*, 1990.
- [5] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long-running activities. In *Proceedings of the 17th VLDB Conference*, september 1990.
- [6] A. Forst, E. Kuhn, and O. Bukhres. General purpose workflow language. In *Distributed and Parallel Databases*, volume 3, April 1995.
- [7] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem. Coordinating Multi-transaction Activities. *Princeton University*, february, 1990.
- [8] N. Krishnakumar and A. Shet. Managing heterogeneous multi-system task to support entire-ide operationswpris. In *Distributed and Parallel Databases*, volume 3, 1995.
- [9] Object Management Group. *Common Object Request Broker: Architecture and Specifications*. Object Management Group, May 1995. revision 2.0.
- [10] R. M. Soley and Chistopher Stone. *Object Management Architecture Guide*. Object Management Group, 3rd edition, 1995.
- [11] H. Wachter, A. Reuter. The Contract Model. In *Transaction Models for Advanced Database Applications*, chapter 7. Elmagarmid Editor, Morgan Kaufmann Publishers, 1992.