Uwe Roth, Ernst-Georg-Haffner, Christoph Meinel:
The Internal Workflow of the Smart-Data-Server
in Proceedings of the IADIS International Conference WWW/Internet 2002 (ICWI 2002), IADIS Press, Lisbon, Portugal,
pp. 572-576, 11, 2002. ISBN: 972-9027-53-6.

THE INTERNAL WORKFLOW OF THE SMART-DATA-SERVER*

Uwe Roth
Insitute of Telematics
Bahnhofstr. 30-32
D-54292 Trier, Germany
roth@ti.fhg.de

Ernst-Georg-Haffner
University of Applied Science
Schneidershof
D-54208 Trier, Germany
haffner@fh-trier.de

Christoph Meinel
Insitute of Telematics
Bahnhofstr. 30-32
D-54292 Trier, Germany
meinel@ti.fhg.de

ABSTRACT

Middle-tier architectures have to fulfill many requirements. These requirements are growing with the different versions of the middle-tier architecture. To allow a flexible extension of the architecture, its design has to be clear and easy so one has not to take care of all the dependencies of the components if a new one is added. In this paper we present the technology of *internal workflow programs (IWP's)*, a specialized flow-chart that makes it easy to add new components to the inner workflow. The definition of IWP's is rather general so that it can be used in different server-architectures. It has been - for instance - implemented in the middleware-platform of the Institute of Telematics, the Smart-Data-Server (SDS) to handle RPC-requests more efficient.

KEYWORDS

Middle-Tier-architecture, internal control flow, internal data flow

1. INTRODUCTION

Middle-tier-architectures have been generally accepted as the intermediators between the offeror of information (e.g. databases) and the consumer of information (every kind of client: e.g. applications, applets, web-servers with servlets/perl). They can be seen as integration platforms to overcome the heterogeneous IT-infrastructures of companies and authorities.

There are different approaches to implement a middle-tier-architecture. The Institute of Telematics Trier, Germany, has developed its own platform that is called *The Smart-Data-Server (SDS)*. The SDS has many interesting aspects which can be read in rhem99a, rhem99b, rem00. Here is a short list:

- Handles RPC-requests
- Component based architecture.
- Components can easily be added.
- Access to the environment of the server with specialized services (Database-Access, Mail).
- Easy adaptation to different operational areas (information about databases, drivers, passwords etc. are stored inside a configuration file).
- A simple feature to built up networks of SDSs to spread the load of the SDSs (information about other SDS is hard-coded into configuration files).

Important is the demand to shorten the time of development of applications. This regards on the one hand the development of the functionality of the middle-tier and on the other hand the simple access of the functionality by the clients.

While developing the SDS1 (SDS version 1.x) platform, our main focus laid on the idea that each component of the SDS has a transparent access to the IT-environment and to shorten the time of developing such components. Target of the SDS2 project (SDS version 2.x) was the improvement of the performance of the SDS. In a first analysis, we extracted a load-balancing component as an extension of the SDS. This component compares the load of several SDSs and routes the requests to the least burdened.

---

* As in: Proc. of the IADIS International Conference WWW/Internet 2002, Lisbon, Portugal, 13-15 November 2002, pp. 572-576

A second field of extension is the request-prediction as a very promising field of exploration [hrem99, hrem00a, hrem00b]. The SDS1 was not developed to bind this new extension into the inner structure. An extension of this structure would only have been possible with disproportionate efforts. In a first step, the inner workflow of the SDS had to be reconsidered to make it easy to insert future extension.

2. CONCEPTION OF THE INNER WORKFLOW

To get a better and flexible design of the sequences inside the SDS a new technology was developed: *inner workflow programs (IWP's)*. It's a structure above the program language (in our case Java) and is processed by a *workflow manager*. This technology seems similar to existing technologies. In the end of the paper we take a look at the differences.

A workflow program consists of one or more *workflow nodes* (with a dedicated start node) and a *workflow net* between those nodes. During the processing of the workflow program, the control flow is transferred among the workflow nodes as a data pool that can be manipulated by the workflow nodes.

The workflow manager is responsible for reminding the inner state of the workflow program (which node is reached, which node has to be reached next, how many subroutines have been called). On the other hand the workflow manager is responsible for executing the java-code, associated with each node.

In the following, each part involved in processing a workflow program is explained.

Data pool

The data pool is a storage for variables. It is very similar to associative arrays or hash-tables. Each workflow node can manipulate the data pool by changing values of variables by adding or removing of variables

Process Node

In fact the work of the workflow program is done inside the process nodes. The process node can maipulate data pool. Depending on the return value of the Process Node (an integer value) the control flow can go different ways inside the workflow net. The Workflow Manager is responsible for this decision.

End node

If an end node has been reached the actual workflow program ends.

An integer-value associated with the end node is returned as the return value to the initiator of the workflow program.

Workflow program

A workflow program itself is a workflow node that can be used inside a workflow program. A workflow program has a dedicated start node (that is not allowed to be a workflow program) where the control flow starts. The control flow is processed inside the workflow program until an end node is reached. The value of the end node decides how the further control flow is processed.

Viewed from the distance the behavior of a workflow program is very similar to the behavior of a process node. Complex program structures are possible when using workflow programs inside workflow programs. Nested programs are possible as well as recursive programs. The risk of implementing never ending runs increases, though.

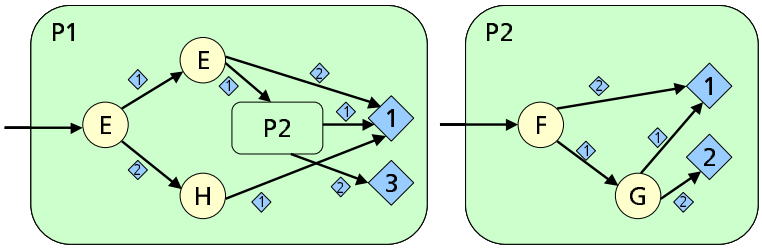
The workflow net

The *control flow* of a workflow program is defined as a *workflow net* between the nodes of the workflow program. The transfer of the control flow to a workflow node has different results.

If the workflow node is a process node, the data pool is transferred to the process node to manipulate it. If it is an end node, the integer-value of the node and the data pool is transferred to the initiator of the workflow. If the workflow node is a program node, the control flow is transferred to the start node of the workflow program (subroutine). Depending on the return-value of the process node, the program node or the end node the workflow manager decides on basis of the workflow net, which node is reached next

3. EXAMPLE OF WORKFLOW PROGRAMS

If the workflow-program of Fig. 1 is started, the workflow-manager looks for the start-node, in this case “E”. The Java-object associated with “E” is executed. Depending on the return-value of “E” the control-flow with the data-pool is moved to the second “E”-object (if the return-value was “1”) or to the “H”-object (if the return-value was “2”). If “P2” is reached during the processing of the program, the start-node of P2 is taken (in this case “F”) and “P2” is processed until one end-node (“1” or “2”) is reached. The value of this end-node is taken to look which direction has to be taken (if “1” go to end-node “1”, if “2” go to end-node “3”).



Examples of possible control flows:
⇒ P1 (⇒ E:1 ⇒ E:2 ⇒ (1)):1
⇒ P1 (⇒ E:2 ⇒ H:1 ⇒ (1)):1
⇒ P1 (⇒ E:1 ⇒ E:1 ⇒
P2 (⇒F:2 ⇒ G:2 ⇒ (2)):2 ⇒ (3)):3

Fig. 1. Example Program

Watching the control flow of a workflow program, the data pool has no meaning, only the return values count. The following notation is used in the examples of Fig. 1:

“⇒ B:1”:the control flow has changed to the process node B. B has returned 1.
“⇒ P(…):4”: the control flow has changed to the workflow program P. P has returned 4.
“(4)”: an end node with the value 4 has been reached.

4. THE RELEVANCE OF THE INNER WORKFLOW FOR THE SDS

To understand the relevance of the new technology for the SDS2-platform one has to take a quick look at the SDS1. The server is divided into three layers (Fig. 2a). A *session layer* to receive requests or to initiate requests. The *function layer* consists of the components of the SDS that contains the functionality of the SDS. This layer tells us, which functionality can be accessed by clients. Both layers access a *service-layer* in which all other components of the SDS are placed in, e.g. services to access the IT-environment (databases, mail, ...).

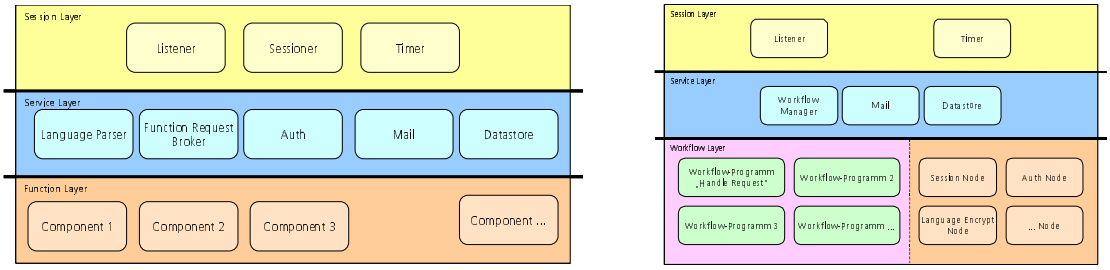


Fig. 2. SDS1 and SDS2

As mentioned in the introduction, the SDS-structure has been implemented to fulfill the need to shorten the time of implementing components. The second approach has its focus in optimizing the internal process to execute the functionality of the components. A first look at Fig. 2b shows only small changes: The function layer is replaced by the workflow layer which consists of workflow programs and workflow nodes. Some components of SDS1 are replaced by workflow nodes and moved to the workflow layer. The application flow of these components/nodes are replaced by specialized workflow programs, which easily be extended in future, if new features should be added to the server.

5. RELATED WORK AND CONCLUSION

The presented workflow programs are very similar to existing notations, such as UML Activity Diagrams [qua00, aey01, mo95], Use Case Maps (UCM) [br96] or others [ehrs00, gbr01, gm01]. But there are some differences. The idea of our workflow programs has its origins in Unix-pipes. It is not possible to split up the workflow path into several parallel workflow paths. These features are possible in Activity Diagrams and UCM’s. Workflow programs are intended to process data streams. In an initial run of the workflow program all components involved get determined and initialized as a pipe.

The processing of the data stream is afterwards processed without any participation of the workflow engine.

The second difference of the workflow programs to existing notations is obviously the workflow engine. In the existing notations nothing is said about how the diagram is processed in real life. The notation is only used to specify the functionality of the system and to built up the structure. But an extension of the resulting structure is different to the extension of our workflow-program. We only have to add the node to the workflow program and tell the workflow manager, which code is assigned with the node.

Talking about workflow normally means to talk about the process of managing different parties involved in a business-process. In our definition the (inner) workflow is a process of involving components in the process of manipulating a data pool. The definition of this process is very abstract and can be used in different contexts (middle-tier-architectures, web-server, ..). In our context we needed the inner workflow with its workflow programs to give a clear structure of our middle-tier-architecture. Other middle-tier-architectures like Enterprise JavaBeans [hae00] or Corba [omg] do not specify the inner workflow from the access of the client to the methods of the objects. Every vendor has its own philosophy to do this.

The complexity of the SDS in the final stage of implementation makes the difficulties obvious, to implement this with the restrictions of the SDS1-platform. The technology of workflow programs gives a flexible structure to extend inner workflow-processes dynamically. The advantage lies in a strict delimitation of the components. Implicit knowledge about dependencies is not possible and has to be defined explicitly at the time of designing the workflow-components. Keeping in mind the visions of the final stage of implementation helps to implement administrable structures.

One aspect that has not been mentioned yet is the liberalization of the workflow-layer for developers of special solutions. In SDS1 it was not possible to implement components outside the function-layer, but the liberalization of the workflow-layer offers new possibilities to the developers e.g. special authorization concepts, only useful for the concrete problem.

REFERENCES

[rhem99a] Roth, Haffner, Engel, Meinel; *The Smart Data Server: A New Kind of Middle-Tier;* As in Proceedings of the Conference on Internet and Multimedia Systems and Applications, IASTED IMSA99, Nassau, Bahamas, 10/1999

[rhem99b] Roth, Haffner, Engel, Meinel; *An Approach to Distributed Functionality: The Smart Data Server (SDS*); As in: Proceedings of the WebNet International Conference, Honolulu, Hawaii, USA, 1999

[rem00] Roth, Engel, Meinel**;** *Improving the Quality of Information-Flow with the Smart Data Server;* As in Proceedings of the 1st International Conference on Internet Computing, IC'2000, Las Vegas, Nevada, USA, 2000

[hrem99] Haffner, Roth, Engel, Meinel; *A Semi Random Prediction Scenario for User Requests;* As in Proceedings of the Asia Pacific International Web Conference, APWEB99, Hong Kong, 9/1999

[hrem00a] Haffner, Roth, Engel, Meinel; *Modeling Time and Document Aging for Request Prediction - One Step further*; As in Proceedings of the Symposium on Applied Computing, ACM SAC2000, Como, Italy, 2000

[hrem00b] Haffner, Roth, Engel, Meinel; *Optimizing Requests for the Smart Data Server;* As in Proceedings of the Conference on Applied Informatics, IASTED AI2000, Innsbruck, Austria, 2000, Onine-Reference: http://httpd.apache.org/docs-2.0/

[ehrs00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. *Efficient algorithms for model checking pushdown systems;* In Proc. of CAV'2000, number 1855 in Lecture Notes in Computer Science, pp 232-247. Springer-Verlag, 2000.

[aey01] R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines; 13th International Conference on Computer-Aided Verification, 2001.

[hae00] R. Monson-Haefel, *Enterprise JavaBeans*, Second Edition, O'Reilly, March 2000,

[br96] R. J. Buhr & R. S. Casselman, *Use Case Maps for Object Oriented Systems*, Formerly published by Prentice Hall, 1996

[mo95] J. Martin & J. Odell, *Object oriented methods: a foundation*, UML Edition, Prentice Hall, 1995

[qua00] Terry Quatrani, *Visual Modeling with Rational Rose 2000 and UML*, 2/e, Addison-Wesley, 2000

[gbr01] Patrice Godefroid, Michael Benedikt and Tom Reps; *Model Checking of Unrestricted Hierarchical State Machines*; Proceedings of ICALP'2001 (28th International Colloquium on Automata, Languages and Programming), Crete, Greece, July 2001. Lecture Notes in Computer Science, vol. 2076, pp 652-666, Springer-Verlag.

[gm01] A. Gal, J. Mylopoulos *Supporting Distributed Autonomous Information Services Using Coordination* International Journal of Cooperative Information Systems, 9(3):255-282, 2001

[omg] OMG; *The Common Object Request Broker Architecture and Specification*; Online-Reference: http://www.omg.org