A VIRTUALIZATION APPROACH FOR REUSING MIDDLEWARE ADAPTERS

Ralf Wagner
University of Stuttgart, IPVS, Germany

Bernhard Mitschang
University of Stuttgart, IPVS, Germany

Keywords: Middleware, adapter, reuse, virtualization, integration.

Abstract: Middleware systems use adapters to integrate remote systems and to provide uniform access to them. Different middleware platforms use different adapter technologies, e.g. the J2EE platform uses J2EE connectors and federated database systems based on the SQL standard use SQL wrappers. However, a middleware platform cannot use adapters of a different middleware platform, e.g. a J2EE application server cannot use an SQL wrapper. Even if an SQL wrapper exists for a remote system that is to be integrated by a J2EE application server, a separate J2EE connector for that remote system has to be written.
Tasks like that occur over and over again and require to invest additional resources where existing IT infrastructure should be reused. Therefore, we propose an approach that allows to reuse existing adapters. Reusing adapters is achieved by means of a virtualization tier that can handle adapters of different types and that provides uniform access to them. This enables middleware platforms to use each others adapters and thereby avoids the costly task of writing new adapters.

1 INTRODUCTION

Middleware systems are commonly used in IT infrastructures. They integrate diverse remote systems and allow applications to uniformly access them. Remote systems are integrated by means of adapters that are plugged into the middleware and that natively access the remote systems. Generally, there is a number of different middleware platforms and adapter technologies. They comprise commercial off-the-shelf (COTS) products as well as research prototypes or in-house integration platforms.

Well-known examples of COTS products are IBM WebSphere Message Broker, Microsoft Biztalk or SAP Netweaver. Often they support industry standards such as the J2EE connector architecture (Sun, 2003), SQL Management of External Data (SQL/MED) (ISO, 2003) or the Web Services Architecture (Booth et al., 2004). Well-known examples of research prototypes are TSIMMIS (Chawathe et al., 1994), Information Manifold (Levy, 1998) or Garlic (Roth and Schwarz, 1997).

A company's tasks and processes change over time and thus its IT systems have to be reengineered to adapt to these changes. This reengineering task is complex and requires to modify, extend or differently arrange and interconnect applications, middleware systems, database system and other back-end systems.

1.1 Example Scenario

Imagine the following example of a typical integration scenario as shown in Figure 1: a J2EE-based decision support system is calculating some enterprise information. It already uses two remote systems, a customer relationship management system (CRM system) and a database system of a human resources application (HR DBS). The enterprise application now has to be extended (dark shaded boxes) to additionally comprise information available from a product management system (PDM system), which is already integrated into a federated DBS by means of a PDM SQL wrapper. Analogously, we now need a PDM J2EE connector to integrate the PDM system into the J2EE application server and to use it with the

Wagner R. and Mitschang B. (2007).
A VIRTUALIZATION APPROACH FOR REUSING MIDDLEWARE ADAPTERS.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 78-85
DOI: 10.5220/0002365600780085
Copyright © SciTePress

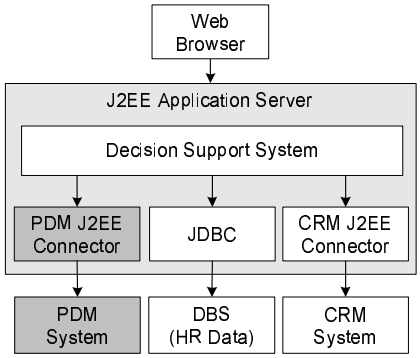

Figure 1: Integration Example: J2EE-Based Enterprise Application.

decision support system.

The problem is that there is quite a number of possible combinations of middleware systems, adapters and remote systems and thus it is very likely that there is no PDM J2EE connector available, but that we have to write a new one.

1.2 Proposed Solution

However, writing a new adapter usually is an expensive, lengthy and error-prone task since it requires substantial knowledge about the middleware, the remote system to be integrated, and the adapter technology to be used for this integration task. The middleware comprises a processing model, a data model, a programming model, an error model, quality of service requirements, etc., which the adapter programmer has to know about and has to deal with. The remote system usually owns a different processing model, data model, programming model, etc., which the adapter programmer also has to deal with. Finally, the adapter technology provides a programming framework that allows to bridge between the middleware and the remote system requiring further knowledge and programming skills for writing an adapter.

In contrast, an existing adapter of course already passed these tasks. Moreover, it has already been tested and maintained in productive use and now works properly. Thus, it would be beneficial to avoid writing a new adapter and to use an existing adapter instead. In our example, we would like to reuse the PDM SQL wrapper instead of writing a new PDM J2EE connector. The problem is that different adapter technologies usually are not compatible, i.e. we cannot use the PDM SQL wrapper with the J2EE application server.

What we need is a mechanism that allows to reuse adapters in different integration scenarios and with different middleware platforms. Therefore, we propose a virtualization tier that uses adapters as their respective middleware platforms would do, but that additionally virtualizes them, i.e. allows to uniformly handle and access the adapters. In Figure 2, the PDM SQL wrapper is deployed into the virtualization tier. Now, the J2EE application server can access the PDM system by reusing the PDM SQL wrapper via the virtualization tier. Hence, there is no need for writing a new PDM J2EE connector.

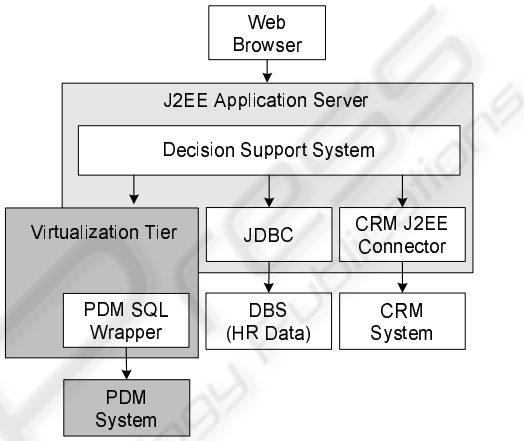

Figure 2: Solution of the Integration Example Using the Virtualization Tier.

Note that the proposed solution cannot just be replaced by adapters that are wrapping other adapters, e.g. a J2EE connector wrapping the PDM SQL wrapper. The reason is that wrapping an adapter would require to provide a complete adapter execution environment, which clearly is middleware functionality since the middleware is executing and handling adapters. Therefore, an “adapter” wrapping another adapter would be more a middleware than an adapter.

Finally, the most important point when considering adapters for wrapping other adapters is that for m middleware platforms on the one hand and n adapters with corresponding remote systems on the other hand we would potentially need $n * m$ adapters that are wrapping the n remote system adapters. This means that each of the m middleware platforms would potentially need n platform-specific adapters for wrapping the n remote system adapters.

In contrast, our virtualization approach reduces the $n * m$ complexity to $n + m$ due to the uniform access that enables a remote system adapter to be used by many middleware platforms. A more comprehensive comparison of the different realization alternatives is discussed in Section 3.2.

In summary, our contribution provides a virtualization approach that allows to reuse adapters by uni-

formly handling and accessing them. Thereby, the $n * m$ complexity of required adapters for n remote systems and m middleware platforms is reduced to $n + m$. The approach is non-invasive, i.e. existing applications and processes of a middleware system are not affected by this integration enhancement. They still work as usual. Finally, the approach provides for a more flexible IT infrastructure that can be more easily adapted to future changes.

In the next section we show the architecture of the virtualization tier and its components. We describe how requests are processed by the virtualization tier and how the deployment of adapters works. Section 3 evaluates the characteristics of the virtualization tier that are necessary to make our approach practically applicable. Section 4 discusses related work and Section 5 concludes the paper.

2 VIRTUALIZATION TIER

The architecture of the virtualization tier (VT) is shown in Figure 3. The VT employs adapter managers that allow to deploy adapters of a specific type, respectively, into the VT and that are responsible for using adapters to access remote systems. For example, a J2EE connector manager would be responsible for deploying and using J2EE connectors in the VT, and an SQL wrapper manager would be responsible for deploying and using SQL wrappers.

The VT uniformly represents data and operations of remote systems as objects that consist of attributes and methods. Access to a VT object results in access to the proper adapter manager that is responsible for using the proper adapter to access the remote data and the remote operations associated with this VT object.

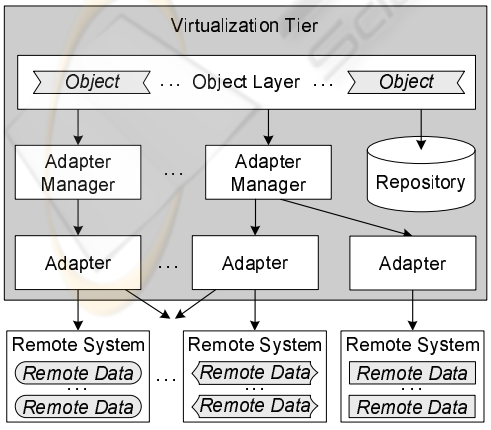

Figure 3: Architecture of the Virtualization Tier.

The object model of the VT has to support the needs of the different adapter technologies, i.e. it has to be able to represent data and operations and it has to be able to execute operations and access single data items as well as whole data sets, preferably in a set-oriented, declarative way. Therefore, we used the ODMG object model (Cattell et al., 2000) to realize the VT object model and we also used the associated set-oriented, declarative object query language (OQL) to realize the VT access language. This allows to represent remote operations as VT object methods and it allows to represent set-oriented queries for remote systems, e.g. SQL queries, as OQL queries in the VT.

A VT object is defined by means of an object configuration consisting of four configuration chapters:

- The *Object Definition Chapter* defines the attributes and methods of a VT object according to the ODMG object definition language (ODL) (Cattell et al., 2000).
- The *Object Information Chapter* defines any information concerning the correlation of a VT object and the associated remote data and remote operations, e.g. which API operation to call or which database table to access in the remote system.
- The *Adapter Information Chapter* defines any information about an adapter. The information is used by the associated adapter manager to deploy and access the adapter properly, e.g. where to find the adapter libraries or which parameters to apply to the adapter.
- The *System Information Chapter* defines any information about a remote system. The information is used by an adapter to properly access the remote system, e.g. authentication information or connection management information.

Configuration chapters are stored in the VT repository. They are retrieved from the repository during runtime and are used in the execution of OQL requests, which is shown in the next section.

2.1 Processing VT Requests

The VT offers an API that allows to submit OQL requests to the VT and to access VT objects. If a VT object is accessed via an OQL request, the VT determines the proper VT object configuration and checks the attributes and methods of the VT object used in the OQL request. If the request is valid, the VT loads the adapter manager associated with the adapter that is responsible for resolving the requested VT object. Then, the VT hands over the request to the adapter manager, which in turn loads the corresponding adapter and issues an adapter technology-specific

request to the adapter. The adapter finally accesses the data or executes the operations in the remote system.

Figure 4 shows how the VT-based solution of the integration example in Figure 2 looks like. The decision support system uses a VT J2EE connector to access VT object *VTScrew*, which is representing information about screws stored in the PDM system (see Section 3.1 for a discussion of the VT J2EE connector).

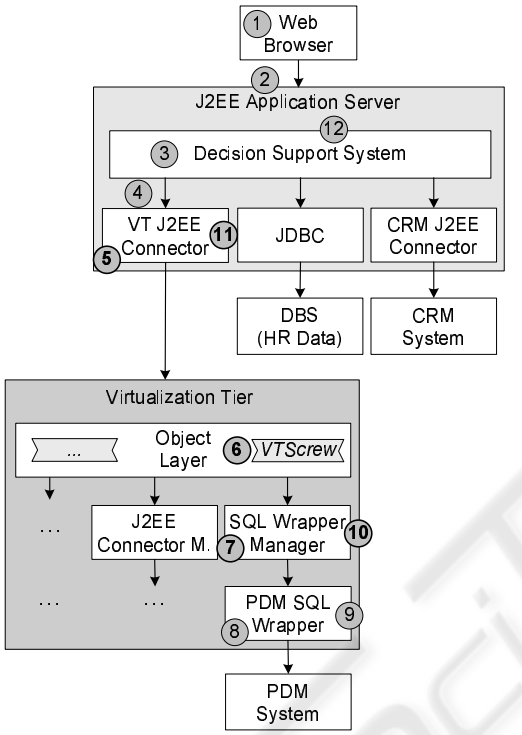

Figure 4: Access to a Remote System by means of the VT.

The benefit of this solution is that the PDM SQL wrapper can be completely reused whereas the conventional solution shown in Figure 1 requires to write a new PDM J2EE connector. In the following we give an example of how the decision support system could access the VT and the PDM system (cf. Figure 4):

1. A user opens a web browser and logs in to the decision support system via a web page.
2. The user wants to access some information about the screws in an air-conditioning machine unit, e.g. material and manufacturer, and submits a corresponding HTML form request to the enterprise application.
3. The enterprise application calls an EJB method to resolve the screw information
4. The EJB in turn issues an interaction request to the VT J2EE connector. Connector interactions are the means by which J2EE connectors are accessed. An interaction object represents a request with input parameters and output parameters. The EJB issues an interaction with the following parameters:
   - Input parameter *access_op* and value *getVTObject* for the access operation.
   - Input parameter *obj_name* and value *VTScrew* for the VT object to access.
   - Input parameter *attr_restr* and value *cunit="air-cond unit"* for further object restrictions, i.e. the unit containing the screws must be an air-conditioning unit.
   - Output parameter *screws*, which is containing a list of identified screws as Java objects.
5. The VT J2EE connector translates the interaction request into the OQL request

   ```
   exists x in VTScrew:
     x.cunit = "air-cond unit"
   ```

   and submits it to the VT.
6. The VT checks the repository for the VT object configuration of *VTScrew* and then identifies the SQL wrapper manager as the responsible adapter manager.
7. The VT hands over the OQL request to the SQL wrapper manager, which in turn translates the OQL request into an SQL query accessing the foreign table containing the desired screw information:

   ```
   SELECT *
   FROM   SCREWS
   WHERE  CUNIT = 'air-cond unit'
   ```

8. The PDM SQL wrapper receives the SQL query and translates it into the corresponding PDM system API call

   ```
   showUnitData("screw", "air-cond unit")
   ```

9. The PDM system returns the requested screw information to the PDM SQL wrapper, which transforms it into SQL data, i.e a table, as the result of the SQL query in 7.
10. The SQL wrapper manager transforms the SQL data into VT object instances according to the *VTScrew* object definition chapter as requested in 5 and returns them to the VT.
11. The VT transfers the VT object instances to the calling VT J2EE connector, which transforms them into Java objects conforming to the *screws* output parameter of the connector interaction in 4.

12. The EJB processes the screw objects and the enterprise application creates a suitable HTML response document answering the initial user request.

Other adapter managers work analogously to the SQL wrapper manager that is submitting an SQL query to an SQL wrapper. For example, a J2EE connector manager would transform an OQL request into a connector interaction with suitable input parameters and output parameters and would execute the interaction on the proper J2EE connector, and a message broker manager would transform an OQL request into a business object request and issue it to a message broker adapter.

Steps 5, 6, 7, 10 and 11 (bold numbers in Figure 4) are introduced by the VT approach. The other steps have to be performed in a conventional integration solution as well; hence, the EJB in step 4 would access a PDM J2EE connector instead of the VT J2EE connector and the PDM J2EE connector would access the PDM system analogous to steps 8 and 9.

2.2 Deployment Process

The skills required to define VT objects and to deploy adapters into the VT comprise knowledge about the VT and the VT object model as well as knowledge about the different involved adapter technologies. Clearly, the complexity of the whole integration task must be reduced to make the VT practically manageable. Therefore, the goal is to divide the deployment process into two steps performed by different persons (as depicted in Figure 5): adapters are deployed in the first step (1) and suitable VT objects are defined in the second step (2).

Deploying an adapter into the VT requires the same deployment information as if the adapter is deployed into its original middleware. A deployer that

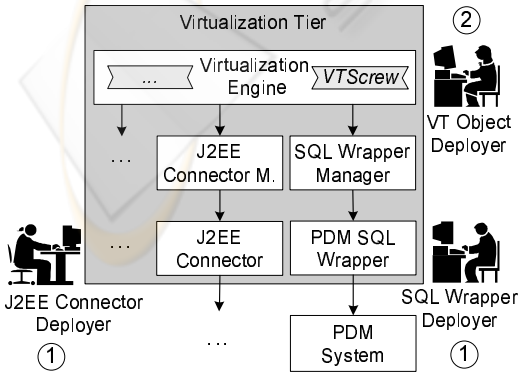

Figure 5: Two-Step Deployment Process.

is responsible for deploying adapters into a middleware (short: adapter deployer; e.g. a J2EE connector deployer or an SQL wrapper deployer) has to be familiar with the middleware, i.e. he knows how to deploy adapters and how to define middleware-specific data and operations that are representing data and operations in the integrated remote systems.

Therefore, the first step of the deployment process is performed by the corresponding adapter deployer. The adapter deployer deploys an adapter into the VT (*adapter information chapter*), correlates a remote system with the adapter (*system information chapter*) and specifies suitable middleware-specific data definitions and operation definitions (*object information chapter*).

For example, if a J2EE connector has to be reused, the responsible J2EE administrator acts as a J2EE connector deployer in the VT scenario. He determines deployment properties of the J2EE connector and of the remote system to access, and he specifies interaction information for executing interactions on the J2EE connector. Or if an SQL wrapper has to be reused, the responsible federated database system administrator acts as an SQL wrapper deployer in the VT scenario. He prepares SQL statements for defining an SQL wrapper, an SQL server and some foreign tables.

The only difference between an adapter deployment in the VT and an adapter deployment in the respective middleware is that the adapter deployer uses the VT deployment GUI for defining configuration chapters instead of the middleware-specific adapter deployment facility.

A deployer of objects in the VT (short: VT object deployer) performs the second step of the deployment process, which requires to define VT objects by means of *object definition chapters*. He correlates the VT object definitions with middleware-specific data definitions and operation definitions specified by adapter deployers during the first step.

The VT object deployer does not have to define the VT objects from scratch since adapter managers comprise functionality that derives default VT object definitions corresponding to the middleware-specific data definitions and operation definitions created during the first step of the deployment process. The VT object deployer uses the generated default VT object definitions as the starting point and only has to customize them for proper usage in the VT.

In this way, the deployment process in the VT is significantly alleviated and becomes practically manageable: The adapter deployer is doing his job as usual. He is only concerned with the middleware-specific part of the VT deployment, i.e. adapter de-

ployment and definition of the middleware-specific data and operations, but he does not need further knowledge about the VT. The VT object deployer is only concerned with the VT-specific part, i.e. definition of VT objects, but does not need to know about the adapter-specific deployment tasks.

3 EVALUATION

There are two further points to be considered when the VT approach has to become practically applicable. First, does the VT affect existing integration processes or middleware infrastructures? And second, what does it cost to provide the VT and its components? Finally, we discuss how the VT provides increased flexibility for IT infrastructures.

3.1 Applicability of the VT

An important aspect of the VT approach is that existing middleware systems do not have to be modified and that the operation of existing applications and processes is not affected. The VT enhances middleware systems in a non-invasive manner offering an additional means of accessing remote systems. The enhancement is achieved by using a middleware's native adapter technology to access the VT, i.e. a VT adapter. For example, the J2EE application server in Figure 4 uses a VT J2EE connector to access the VT.

Figure 6 abstractly shows how existing and new middleware applications coexist. On the one hand, existing applications only use adapters of the middleware that are directly accessing remote systems. On the other hand, new applications can additionally include the VT in their processing.

Another point is that the VT requires a suitable adapter manager to deploy and use adapters of a certain type, e.g. an SQL wrapper manager is required to deploy and use SQL wrappers in the VT, a J2EE connector manager is required to deploy and use J2EE connectors, etc. If an adapter not yet supported by the VT is to be reused, a suitable adapter manager for that adapter type must be written. Writing a new adapter manager is at least as costly as writing a new adapter. But if it is necessary to write several new adapters to integrate some remote systems and if this can be avoided by reusing one or more adapters with the new adapter manager, it is worth writing this adapter manager. Writing the adapter manager has to be done only once, but allows to use any adapter of that type in the VT.

Similar considerations hold for middleware platforms that want to access the VT since such a middleware platform requires a suitable adapter to enable VT access, i.e. a VT adapter. For example, a J2EE application server requires a VT J2EE connector to access the VT and a federated DBS requires a VT SQL wrapper. If such an adapter is not available for a middleware platform, it must be written. But if it is necessary to write a couple of new adapters for that middleware platform to integrate some remote systems and if this can be avoided by reusing existing adapters in the VT, it is worth writing the VT adapter. Writing the VT adapter has to be done only once, but allows the middleware platform to use any adapter that is deployed in the VT.

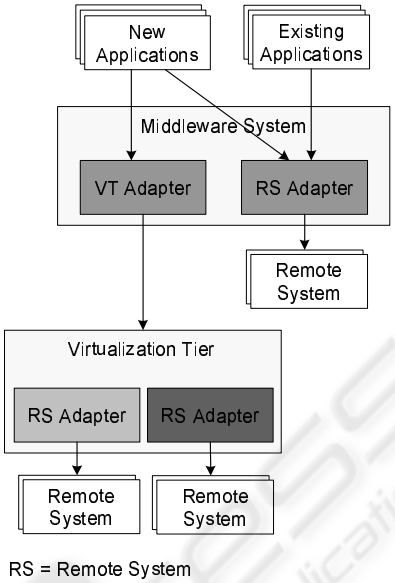

Figure 6: Enhancing Middleware Systems in a Non-Invasive Manner (Abstract View).

3.2 Increased Flexibility

An additional benefit of the VT approach is that it increases the flexibility of an IT infrastructure since future changes and requirements concerning such integration tasks can be solved with the VT again. The more middleware systems in an IT infrastructure use the VT and the more adapters are deployed into the VT, the more likely is it that a desired combination of middleware and remote system is already available and that it can be used without writing a new adapter.

Abstractly seen, the VT can be considered as a middleware multiplexer that allows a middleware system to access any adapter that is deployed in the VT. If m middleware systems have to access n remote systems *without* using the VT, we would potentially need $n * m$ adapters as shown in Figure 7. If we use the VT,

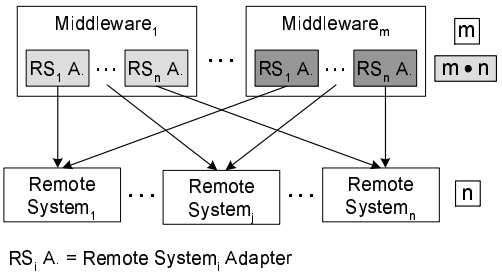

Figure 7: Conventional Integration Approach: Potentially $m * n$ Adapters Required.

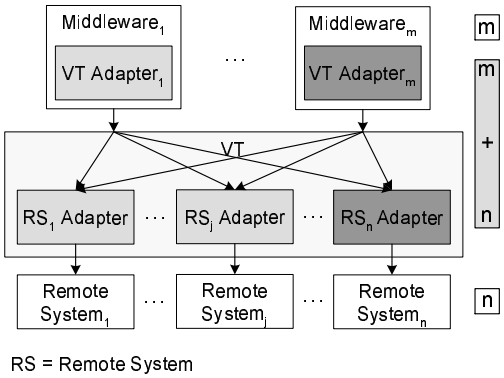

Figure 8: Virtualization Tier: Multiplexer Characteristic; only $m + n$ Adapters Required.

this complexity is reduced to $m + n$ adapters as shown in Figure 8 (m adapters for accessing the VT and n adapters for accessing the remote systems).

In other words: a conventional integration approach relying on a specific middleware system and a specific adapter technology needs a specific adapter to integrate a remote system whereas the VT allows to use the same middleware system, but any adapter that is available for that remote system.

4 RELATED WORK

The problem of incompatible adapter technologies is inherently given with the usage of different middleware platforms and different integration technologies. To the extent of our knowledge systematic reuse of adapters is not possible so far. However, the need for flexibly dealing with adapters is a general requirement and therefore there has already been done substantial work on how to ease writing adapters.

Adapter technologies usually come with adapter frameworks, which at least provide commonly used adapter functionality as code libraries. For example, the WebSphere Business Integration Adapters (IBM, 2004) divide an adapter into two parts. The so-called *connector framework* that contains the functionality common to every adapter, e.g. communicating with the integration broker. And the so-called *application-specific component* that contains the functionality that is different for each adapter, e.g. calling the respective remote system API.

More sophisticated adapter technologies comprise complex system-internal interactions between middleware and adapter, e.g. see (Booth et al., 2004; ISO, 2003; Sun, 2003). For example, J2EE connectors (Sun, 2003) rely on functionality residing in the application server. The application server provides functionality, e.g. transaction management or connection handling, that is commonly available to every connector by means of so-called system contracts.

Advanced approaches aim at providing a high-level specification of an adapter's functionality so that the desired adapter code is generated or existing adapter libraries are suitably parameterized, e.g. see (Ashish and Knoblock, 1997; Baru et al., 1999; Gruser et al., 1998; Hammer et al., 1997; Liu et al., 2000; Raposo et al., 2002). For example, the TSIMMIS project (Hammer et al., 1997) puts common parts of a wrapper implementation into a common library used by any TSIMMIS wrapper. The library is parameterized for each wrapper by high-level, declarative rules that determine what queries can be executed by the wrapped remote system, what the answers look like and how the transformation between the queries and the data in TSIMMIS on the one hand and the remote queries and remote data on the other hand is performed.

Adapter frameworks and adapter generation approaches inherently can handle only that parts of an adapter that are common to all adapters or that are at least similar for a group of adapters. However, the heterogeneity of remote systems would require generation approaches to flexibly deal with different access paradigms, request processing styles, data structures, data models, programming languages, APIs, etc. But this complex task cannot be solved just by parameterizing a library or by specifying a set of declarative, high-level rules to generate the necessary code.

Adapter generation only works if the targeted remote systems are restricted to a specific type so that most characteristics and properties are known in advance and can be considered in common libraries, rule sets or high-level scripting languages. For example, the generation approaches in (Ashish and Knoblock, 1997; Gruser et al., 1998; Liu et al., 2000; Raposo et al., 2002) are targeted at web information sources, i.e. primarily HTML pages. Other approaches such as in (Baru et al., 1999; Hammer et al., 1997; Pan

et al., 2000) support more than one remote system type based on semi-automatic adapter generation procedures, but each remote system type inherently requires a separate adapter library, i.e. a different adapter.

The recent evolution of universal metadata-driven generation approaches such as OMG's model-driven architecture (MDA) (Miller and Mukerji, 2003) could lead to techniques applicable to the generation of adapters, too. But there are no results for this kind of problem so far nor do we see any progress for that in the next time.

In contrast, our virtualization approach provides a practical solution for dynamically reusing adapters without affecting existing applications and without modifying existing middleware systems.

5 CONCLUSION

The use of different middleware platforms and different adapter technologies leads to repeating programming efforts, i.e. writing adapters. Writing a new adapter is a costly task. Therefore, we proposed a virtualization approach for reusing existing adapters. The VT virtualizes adapters by uniformly handling and accessing them and thereby reduces the current complexity of $n * m$ adapters for m middleware platforms and n remote systems to $n + m$.

There are crucial points that decide about the applicability of the VT approach. The first is the reduction of the deployment process complexity by dividing the deployment process into two steps performed by separate deployers. Adapter deployers are concerned with their respective adapter technology only, and the knowledge about the different adapter technologies that is required by a VT object deployer is reduced to a minimum.

The other point is that the VT can be smoothly used with existing IT infrastructures: their operation is not affected by the VT. Additionally, the VT approach provides for more flexibility in integration tasks. It leverages existing IT resources the better the more middleware systems use the VT and the more adapters are reused by the VT.

Currently, we are finishing our prototype and prepare for extensive experiments to support our approach in kinds of quantitative results, which is the last cornerstone of the VT evaluation. The experiments will include different ways of realizing adapter managers and they will consider different adapters, adapter types and remote systems as well as different workloads.

REFERENCES

Ashish, N. and Knoblock, C. A. (1997). Semi-Automatic Wrapper Generation for Internet Information Sources. In *COOPIS '97*.

Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., and Chu, V. (1999). XML-Based Information Mediation with MIX. In *SIGMOD '99*.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., editors (2004). *Web Services Architecture*. World Wide Web Consortium. W3C Working Group Note.

Cattell, R. G. G., Barry, D. K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., and Velez, F., editors (2000). *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann.

Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. (1994). The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *16th Meeting of the Information Processing Society of Japan*.

Gruser, J.-R., Raschid, L., Vidal, M. E., and Bright, L. (1998). Wrapper Generation for Web Accessible Data Sources. In *COOPIS '98*.

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., and Vassalos, V. (1997). Template-Based Wrappers in the TSIMMIS System. In *SIGMOD '97*.

IBM (2004). *A Technical Introduction to Adapters*. International Business Machines Corporation. WBIA Adapter Framework, V2.4.

ISO (2003). *Information technology – Database languages – SQL – Part 9: Management of External Data (SQL/MED)*. International Organization for Standardization, 2nd edition. ISO/IEC 9075-9:2003 Published Standard.

Levy, A. Y. (1998). The Information Manifold Approach to Data Integration. *IEEE Intelligent Systems*, 13(5).

Liu, L., Pu, C., and Han, W. (2000). XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *ICDE '00*.

Miller, J. and Mukerji, J., editors (2003). *MDA Guide Version 1.0.1*. Object Management Group Inc.

Pan, A., Raposo, J., Álvarez, M., Montoto, P., Orjales, V., Ardao, J. H. L., Molano, A., and Viña, Á. (2000). The Denodo Data Integration Platform. In *VLDB '00*.

Raposo, J., Pan, A., Álvarez, M., Hidalgo, J., and Viña, Á. (2002). The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In *DEXA '02*.

Roth, M. T. and Schwarz, P. M. (1997). Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *VLDB '97*.

Sun (2003). *J2EE Connector Architecture Specification, Version 1.5*. Sun Microsystems Inc. Final Release.