Approaching a Methodology for Designing Composite Applications Integrating Legacy Applications Using an Architectural Framework

Helge Hofmeister*
BASF IT Services
Brussels, Belgium
Email: helge.hofmeister@basf-it-services.com

and

Guido Wirtz
Distributed and Mobile Systems Group
Otto-Friedrich Universität Bamberg
Bamberg, Germany
Email: guido.wirtz@wiai.uni-bamberg.de

Abstract. In this paper, an approach toward a problem-oriented design methodology for building composite applications is proposed. This methodology allows for a business process based design that focuses on re-using both existent enterprise services and legacy functionality. The methodology takes a reference architecture into account that describes how composite applications can be structured by means of layers and which analysis and design patterns can be identified and applied on these layers. The presented methodology aligns its phases on the architecture's single layers and on the dependencies that do exist between the patterns of the single layers. These patterns are described as well.

* Contact

1 Introduction

Service oriented architecture (SOA) is an emerging spirit that promises to allow agile software development by re-using existent functionality. The software applications that re-use functionality are called composite applications.
Inherent aspects of a SOA that should allow for such a re-assembling of functionality are transparent distribution of functional entities, modularization and facilitated (re-)ordering of functions. In this world, distributed functions are called services. Transparent distribution and modularization are of course existent and exploited for quite a time — even if the modularization of services in terms of granularity is different e.g. to object orientation. So the promise of this approach lies in the facilitated composition of services that comes with arising standards as [1] or [2] that are orchestration mechanisms for web services. Together with the SOA and the first service orchestration languages, people started thinking of applying some sort of business oriented protocol (such as business processes) to basic services and having automated support for the business process right away. We believe that it requires more than technical standards to make this dream come true. Furthermore, requirement analysis as well as the development should be structured by reference architectures and development methodologies.

The quality of a system's design is partially determined by the quality of the basic components' design. Thus, it is important not only to compose services but also to compose well-designed services. This is why there do exist already sophisticated definition processes (such as [3]) or quality metrics (such as [4]) that support the grouping of functionality into services. But despite of the advantages of these approaches, services are, no matter how good they are designed in terms of functional or informational cohesion or the degree of inter-service coupling, by definition distributed across organizational borders, defined by different people and executed by different agents. So it is not realistic to expect well-designed services that are easily to be orchestrated by a business process that is translated into a service orchestration language. Although we consider the SOA approach and all the benefits of its agility as promising, we believe that business requirements that are expressed by business processes should not be restrained by the incompatibility of services. As we derived our initial principles from the analysis of case studies, we consider the heterogenity of application landscapes as a crucial point – even if the landscape is wrapped using services. Hence, our aim is to provide mechanisms that allow for a requirement engineering that is not constrained by the definition of existing services.

After an overview of related work in section 2, the reference architecture and its relevant patterns are outlined in section 3. Afterwards the design phases of the design methodology are presenteded.

2 Related Work

There do exist a lot of software development methodologies, such as the Rational Unified Process [5] or — still notably — the waterfall model [6]. However, these methodologies describe software development methodologies while we focus here only on the aspect of designing the blueprint for the software.
The workflow patterns presented by van der Aalst et al. [7], the workflow data patterns by Russel at al. [8], the service interaction patterns by Barrows et al. [9] as well as the enterprise integration patterns by Hohpe and Woolf [10] are the patterns that can identified at the architecture's layers.
Although not working with patterns but in terms service design, the work of Reijers [4] and Feuerlicht [3] provide means for constructing services. The main difference between the presented work and these service definition methodologies is that the presented methodology focuses on re-using functionality of an existent application landscape rather than adding new services.
A comprehensive introduction to service oriented architectures is given in [11].
Multi-layered reference architectures are of course used as well in the area of service oriented design. Notably is the work of Decker [12] who also describes an intermediate layer between business processes and application services that align process and IT. Whilst this work focuses on the semantical gap between processes and services, this is one among several aspects one of the reference architecture used here.
In the area of mapping workflow descriptions to each other Dehnert and van der Aalst developed an approach to map business process descriptions onto workflow descriptions using petri-nets [13]. This work is complementary to our work as they describe how to derive the fourth level of business processes that are executable by means of workflows. A methodology for migrating legacy applications is given in [14]. The butterfly methodology is not focusing on integrating legacy systems but on migrating them while operating the two worlds in parallel. Anyway, some aspects of this approach such as the Chrysaliser for data migration decribe similar concepts as the propsed reference architecture.

3 Architectural Framework for Composite Applications

According to Linthicum, Business Process Integration Oriented Application Integration (BPIOAI) provides another layer on-top of existent system integration concepts such as information oriented application integration (IOAI) or service-oriented application integration (SOAI) (cf. [15]). This means, that system-integration focused technologies such as e.g. JMS messaging (for IOAI) or HTTP-based web services (for SOAI) are controlled by a top-level orchestration layer that implements the business logic. Often, to this business-process implementation, it is referred to as composite application. While the composite application implements the business logic, the used integration technologies solely provide the means for calling application systems that in turn provide the logic that is orchestrated by the composite application. To this part, not implementing any

business logic, it is referred to as coupling system.
In this chapter we present a reference architecture for composite applications that leverages the development methodology. The framework is structured by five layers that provide different abstraction levels for composite applications. An overview of the architecture is given in figure 1.

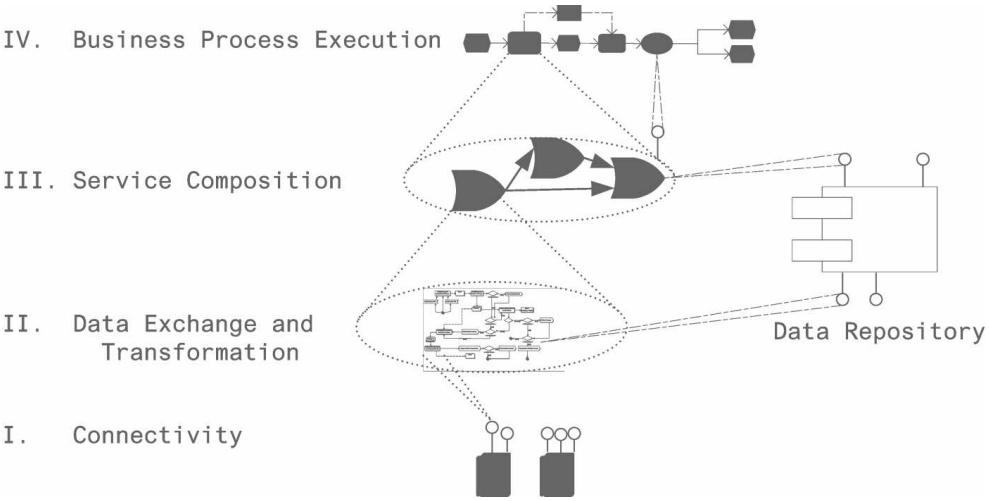

Fig. 1. Architectural Layers

3.1 Layer Zero - Legacy Application Systems

Composite applications reorganize services as they are provided by application systems in order to meet specific requirements. The application system's services are often predefined by the system's vendor. Alternatively, it might be required to define new services of an application system or to develop new agents that provide new services. As a prerequisite, service oriented architectures need to rely on a common protocol that is shared both by service consumers and providers (cf. eg [15][p. 218]). How application services are technically connected to service consumers, is described at the subsequent layer.

3.2 Layer One - Connectivity

In order to allow the usage of application system functionality in composite applications, this functionality needs to be exposed in a common way. This exposure is described at this layer of connectivity. Here, multiple state transitions of the connected application system are exposed as services. So this layer provides connectivity in between the legacy application system and the composite application by homogenizing the protocol that is used to call functions. From the

composite application point of view, this layer provides the application services. Since this layer solely assures connectivity, the actual data format at this layer is still dependent to the connected systems. The exchange and conversion of data is provided at the subsequent layer.

3.3 Layer Two - Data Exchange and Data Transformation

This layer is dedicated to deal with technical complexity by integrating heterogeneous application systems and providing homogeneous interfaces to more high-level functionality. Since we do not mix up business logic with data exchange and transformation functionality here, the probability for re-using functionality from this layer is higher. In turn, subsequent layers do not need to deal with this sort of technical issues. Additionally this layer unifies the data format of the connected application systems to a canonical data format (cf. [10][pp. 355-360] or [16]). After unification of the data, the data is stored into a data repository that is part of our reference architecture as well. This data repository provides the context to the processes that are defined at this and higher layers. Using this context, all services of a composite application can access and exchange data. Additionally this layer provides means for validity checking of data in terms of syntax and semantics as well as error handling procedures that need to be invoked whenever errors occur on this layer[1]. Hence, it provides the functionality that encapsulates the actual communication semantics with the connected systems. This means for instance that acknowledgements for asynchronous calls are transparently handled by this layer. The technical routing is performed here as well.
All the functionality that is provided by this layer is encapsulated in so-called integration services (IS). These services are in turn orchestrated to two different integration processes. One providing data to the upper layers — the integration in-flow (IIF) — and one for publishing data from upper layers to the connected legacy systems. The latter process is called integration out-flow (IOF).
For a more detailed description of the integration services, integration flows and patterns at this layer you might consider [17].

3.4 Layer Three - Service Coordination

From a top-down perspective the integration flows provide together with the connectivity services at layer one two standardized services for calling services at or consuming services from underlying application systems. This is irrespectively of communication or computational semantics and provides homogeneous data access as well.
The functionality that is provided by these services is nevertheless determined by the functionality offered by the application systems. It might be necessary to

[1] The error handling at this layer basically covers support procedures that need to be initiated whenever errors occur (mostly human errors are the cause for this sort of errors).

aggregate the application services to more problem-oriented services (enterprise services). At this layer we see the coordination of two to n application services with a flow in order to form the enterprise services out of application services.
Conceptually, this coordination layer is recursive. This means that an aggregated service that is composed at this layer might be orchestrated together with services from this or lower layers to form other high-level services at this layer.
Besides for the sake of bringing together requirements and application services, coordination at this layer can also be applied to integrate relevant choreographies into the implementation of a business task.
Ensuring proper state transitions is also matter to this layer's coordination. Rather short-term transactions fulfilling the ACID properties as well as long-running transactions might be required by a composite application.
According to Grefen [18], the transactional coordination consists in the so-called *local transactions* with ACID properties and one the *global transactions* with relaxed transactional properties. The latter one uses ACID transactions as black-boxed functionality to form long-term global transactions.
In our framework we meet these ideas by assigning local transactions as being encapsulated at this layer into the compositions of the enterprise services that are exposed by this layer to the business process layer. So the local transaction layer is completely located at our coordination layer and the coordination protocol (such as the 2-phase commit) is fully implemented here.
Meeting the long-term characteristics of global transaction, according to Grefen et al. isolation is relaxed by publishing intermediated results to the global context. Atomicity is relaxed by introducing compensation transactions. Both context publication and compensation transactions are local transactions. Safepoints are local transactions that are marked by this special property of being a safepoint. Thus, the fundamental support for global transactions is formed by local transactions.
[18] proposes as well a way of specifying transactional properties (such as the safe point properties for local transactions) and an execution model that supports global transactions based on these specifications.

The service choreography that is incorporated into this layer, describes how the external services and the service that is formed by the layer interact. In order to analyze the interactions we used the service interaction patterns by Barrows et al. [9].

Decker introduces a so-called process support layer that deals with various aspects of incompatibility between business process tasks and application services [12]. To a certain extent some of these ideas are supportive in terms of designing the third layer. Anyway, the process support layer cannot be mapped directly to our layer of service coordination but most of the referenced patterns are relevant here as well. These are the patterns for granularity problems as well as the patterns for interdependency problems.

The third purpose of the service coordination layer is to leverage transactions between multiple application services. As transactional interaction is a requirement, these requirements have to be included into the design at this

layer. In order to capture this design knowledge as well by the means of patterns we introduce two design patterns as they are informally included in Grefen's work [18]: Local Transaction Composition (LTC) and Global Transaction Composition (GTC).

In order to support these two transactional patterns, we require three properties for services or transactional compositions as mandatory. These are: Safepoint, Idempotent and Compensation. Note that the *Safepoint* property is unary whilst *Idempotent* and *Compensation* are binary properties between transaction compositions. These properties need to be assigned to the single services of a GTC in order to allow the calculation of GTCs' compensations.
While an LTC is a black box and commits changes to the context of a workflow after a successful completion, GTCs are more interactive in terms of context updates. In order to describe the interactions with the context, the workflow data patterns from [8] are referenced as design patterns at this layer as well.
In this context the data visibility patterns are used to determine how single services of one transaction compositions share their data. The data exchange between transaction compositions is described by data interaction patterns. The data transfer patterns can describe the mechanism of how contexts that are only accessible by a transaction composition can be made available to other services.

3.5 Layer Four - Business Processes

The 5th layer is the layer where business processes are to be executed using workflow models. The actual functions of the according business processes are realized as services that are exposed by the lower layers. At this layer it is of importance important how process branching based on certain indicators (like states) might be performed. Thus, the process environment needs to have access to the actual context of the process. The data transfer between application services and the actual context is realized by the Data Exchange and Data Transformation Layer. The visibility of data is controlled by the Service Coordination layer and transactional properties are incorporated into the process environment this way. Access to the process context is realized by the services that are provided by the coordination layer.

In order to analyze, design and automate business processes with regards to composite applications, the control as well as the data perspective are important at this layer (cf. [7]). This is because the sequence of underlying tasks that are represented by services is determined at this layer.
For the perspective of control flows there exist a catalogue of so-called workflow patterns. The work of van der Aalst et al. consists in a set of patterns that are distinguished into basic control flow patterns, advanced branching and synchronization patterns, patterns involving multiple instances, state-based patterns and cancellation patterns.
For the data perspective there exists a broad set of patterns as well. [8] distinguishes the data patterns into patterns for data visibility, data interaction, data transfer mechanisms and data based routing. Solely the category of data based routing pattern is relevant for this layer as these patterns describe how data

needs to be accessed in order to guarantee the execution of workflows. As other aspects are not covered by this layer, are the related patterns also relevant for other layers.
All these patterns are artifacts that can be identified within a workflow description[2]. Thus, we consider the mentioned sets of patterns as analysis patterns that can be used in later phases on lower layers in order to support the design ibidem.

4 A Methodology to Design Composite Applications

The methodology is closely related with the layers and patterns that have been introduced in the previous section. Thus, describing the methodology basically consists of discussing the phases that are required to step through that reference architecture.

4.1 Business Process Modelling and Requirements Engineering

First of all, our approach is very business process centered. Thus, it is absolutely required to model the relevant business process using a methodology of choice. Important is that the modelled business process describes not only the control flow perspective. Furthermore, it is required to also describe the data, organizational and operational perspective[3]. Having this process description as an input, the phases that are described below can be applied.

Enterprise Service Matching Having described the functional requirements of a to-be built composite application by expressing the control and data flow perspective of an business process, the single process steps — usually referred to as functions — need to be matched with services. In order to support the execution of these functions, the operational perspective needs to be checked in terms of finding application systems that might already expose suitable services. *Suitable* in that case means that the service offers operations that *match* the functional requirement. This requirement is usually expressed in a non-formal (usually verbal) way in terms of required functionality, and in terms of the process's data perspective. The data perspective here describes which input and output format are to be sent/received by the identified service.[4]
Whenever there is no suitable service in the organization's service repository,

[2] Of course, they are also useful for designing engine-specific workflow descriptions out of business process descriptions.
[3] Note that the operational perspective is — especially for new systems — often incomplete at the time of requirements engineering.
[4] Even if a lot of work has provided means for matching services, even dynamically at runtime, we do not consider these possibilities yet, since large organizations usually lack a formal description of their legacy systems' functionality. The same is currently true for more recently built service repositories.

but the operational perspective of the business process refers to one or more application services, the functionality of these application systems need to be analyzed in terms of finding functional areas that might be appropriate for realizing the required functionality. In large organizations, usually a fraction of the required functionality is already existent in legacy systems.
Anyway, sometimes there might not exist any application system at all, and the business process function needs to be implemented from scratch.

The artifact of this phase is either the enterprise service identified in the repository, the functional areas of relevant application systems that might be used to implement the enterprise service or the information that a new service is required to be developed.

Data Interaction Analysis The next step is to analyze the data interaction as it is roughly described in the data perspective and in the control flow perspective of the business process. This can be done by identifying data interaction patterns' (cf. [8]) data-based routing patterns within the process. This information is required in order to identify which data needs to be accessed by the process execution layer in order to determine the appropriate process branch.

The artifacts of this phase are the identified patterns as well as the relevant part of the data-model the pattern operates on as well as specific attributed of the patterns such as target values or branching conditions.

Transactional Property Identification Identifying transactional requirements in the business process is a rather difficult task. Therefore, several iterations are required to gather a complete picture. As a first step, it is required to express the already known transactional spheres in the business process. These might either be a set of functions[5] that need to be grouped into one ACID transaction, or more probable, sets of functions that need to be grouped into *global transactions* with relaxed ACID properties.

The artifacts of this phase are the subgraphs of the business process and the information they should be handled as *local* or *global* transactions.

Definition of Non-Functional Requirements In the next phase, non-functional requirements such as required response times have to be collected for the single functions of the business process. An artifact of this phase could be e.g. a catalogue as it is described in [19].

Service Composition This phase marks the transition from the business process layer to the service composition layer. Here it is identified whether certain business protocols are required to be respected while supporting single business process functions. The part of the protocol that is required to be implemented by the actual organization has to be expressed by service interaction patterns

[5] This set might also solely contain a single function.

(cf. [9]). Eventually, other services than the already identified are involved in such a protocol. This phase finalizes the description of the service coordination layer's interactions.

The artifacts of this phase are the references to the involved services as well as the description of their interactions by the means of the service interaction patterns.

Application Service Determination At this stage the design decision is made how the (eventually) identified lack of enterprise service can be overcome. For that sake the application systems that have been identified during the *enterprise service matching*-phase needs to be refined in terms what functionality can be combined in order to form the required enterprise service. For that sake, potential functions of legacy systems, service interaction patterns and the patterns of [12], need to be used in order to describe the interaction between the legacy systems. In order to not modify standard software, also functions from the same legacy system might be required to (re-)composed by the interaction described in this phase.

The artifact of this phase are the detailed description of the legacy functions as well as the description of their composition by means of the named patterns.

Transactional Composition Analysis Taking the information from the first transactional analysis phase into account, this phase uses this information together with the complete description of the service coordination layer in order to complete the picture of transactional properties.
Again, this involves the identification of workflow sub-graphs in terms of them being *local* or *global transactions*. Additionally, single steps or complete *transactions* need to be categorized. This involves assigning the properties *Safepoint*, *Idempotent* and *Compensation*. The *Safepoint* property is unary and therefore only to be assigned to a single transaction, *whilst Idempotent* and *Compensation* are binary properties between *transactions*. These properties can later on support the runtime to calculate compensations which is necessary to allow for long-running transactions with relaxed ACID properties.

The artifacts of this phase are both the identification of *transactions* in terms of their composition sub-graph as well as their properties.

Top-Down Data Repository Design With the yet produced artifacts, it is possible to determine which data in the data repository has to be accessed when and by which component. Additionally, it is determined, how these accesses need to be protected in terms of transactional properties. As a result, in this phase it is possible to describe the interaction with the data repository that establishes the context for the composite applications from a top-down perspective.

The artifacts of this phase is the description of operations that need to be performed against the data repository. This includes the description of the interaction as well as the possibly necessary design of new interfaces and wrappers (especially facades for the transactional access to data entities might be required.

Data Exchange and Data Transformation Definition Having described the service coordination layer in total, the produced artifacts can be used in order to design the layer of data exchange and data interaction. For that sake the service interaction patterns can be applied in order to constrain the patterns that can be used in the design of this layer. Tables 1 and 2 describe how the service interaction patterns constrain this design. Crosses (X) indicate mapping between patterns, swung dashes (˜) indicate that the according pattern cannot be used.

Interaction Pattern	IIF	Sync	Async	Event	Create MI	Select Cons	Polling Cons	Validity	Ack Valid	Refuse Inval	Receiver	Ack after Rec	Msg Seq	Splitter	Aggreg	Resequencer	Store Context	Transformator	Ack aft Forw
1. Send	~																		
blocking																			
non-blocking																			
Party unknown																			
Party known																			
2. Receive	X			X							X						X		
Reliable Receiver																			X
Discarding Msg						X													
3. Send/Receive	X			X		X					X								
PartyKnown																			
PartyUnknown																	X		
Blocking		X																	
1 Continuation																			
2 Continuations					X														
4. Racing Incoming Msg	X			X							X								
different msg types					X														
different processes					X														
discard 2nd message						X													
ranking						X													
non-deterministic						X													
5. One-to-many send	~																		
nr of parties known																			
nr of parties unknown																			
reliable delivery																			
6. One-from-many receive	X			X		X					X				X				
7. One-to-many send/rec	X			X		X					X								
nr of parties known																			
nr of parties unknown																			
reliable delivery																			X
8. Multi Responses	X			X							X								
unique message type																			
different message types					X	X													
stop notification																			
9. Contingent Requests	X				X	X w TimOut					X								
10. Atmoic Multicast ntf	~																		
11. Request with Referral	~																		
refer to single party																			
refer to multiple parties																			
a priori known parties																			
unkwon redirect parties																			
12. Relayed Request	~																		
13. Dynamic Routing	~																		

Table 1. Pattern Mapping Concerning the IIF

Applying these dependencies leads to a pre-selection respectively to discarding of several patterns of the data exchange and transformation layer.
In order to complete the design, the integration flows that have been identified as being relevant have to be checked in terms of which of its patterns is required. Additionally, all parameters coming along with these patterns have to be described as well. The non-functional requirements are used for some decisions at this point as they further constrain the decisions.

The artifact of this phase is the complete description of the required integration in-flows as well as the description of the integration outflows. Note, that this also defines in detail which legacy systems are integrated into the composite application by which means.

Final Design of the Data Repository with Bottom-Up Aspects Having specified the second layer — especially the data that flows into/out of the

Interaction Pattern	I0F	Sync	Async	Fetch Context	Pre-Transform	Router	Content Based	Dynamic	Rec. List	Transformator	Updater	Resequencer	Sequence	Aggregator	Cmd Msg	Doc Msg	Event Msg	Ack If Succ.	Ref. On Fail	Error Handling
1. Send	X										X									
blocking		X																		
non-blocking			X																	
Party unknown								X												
Party known																				
2. Receive	~																			
Reliable Receiver																				
Discarding Msg																				
3. Send/Receive	X										X									
PartyKnown																				
PartyUnknown				X				X												
Blocking		X																		
1 Continuation																				
2 Continuations																				
4. Racing Incoming Msg	~																			
different msg types																				
different processes																				
discard 2nd message																				
ranking																				
non-deterministic																				
5. One-to-many send	X										X		X							
nr of parties known																				
nr of parties unknown								X												
reliable delivery																				X
6. One-from-many receive	~																			
7. One-to-many send/rec	X					X					X									
nr of parties known									X											
nr of parties unknown								X												
reliable delivery																		X	X	
8. Multi Responses	X										X									
unique message type																				
different message types																				
stop notification																				
9. Contingent Requests	X										X									
10. Atmoic Multicast ntf	X	X									X							X	X	X
11. Request with Referral	X										X						~			
refer to single party																				
refer to multiple parties						X					X									
a priori known parties									X											
unkwon redirect parties								X												
12. Relayed Request	X										X				X	X				
13. Dynamic Routing	X																			

Table 2. Pattern Mapping Concerning the I0F

Store/Fetch Canonical Data integration services (cf. [17]), allows for finalizing the data repositories design from a bottom-up point of view.

As for the top-down iteration, the artifacts of this phase is the description of operations that need to be performed against the data repository. This includes the description of the interaction as well as the possibly necessary design of new interfaces and wrappers (especially facades for the transactional access to data entities might be required.

Connectivity Check In order to connect the identified legacy systems to the composite applications, this phase checks whether the connectivity (layer one of our reference architecture) is sufficient. This is quite straight-forward since only the idioms for the *Event*, the *Receiver* as well as for the *Updater* service need to be supported.

The artifacts of this phase is the information whether the supplied connectors are sufficient in terms of these services and idioms. If the support for the identified idioms by means of connectors is not sufficient, a dedicated project might be launched that addresses the identified issues.

Service Design As this methodology focuses on re-using legacy application systems, we do not detail the phase(s) that are required whenever functionality can not be built by (re-)assembling existent functionality. Due to the immense complexity of this topic, only as an place-holder this methodology incorporates a design-phase that has to design missing services.

The artifact of this phase(s) are the design for the implementation of new enterprise or application services.

5 Summary and Outlook

We have presented an approach how a reference architecture can be used to formulate a design methodology that builds on-top of legacy applications. As we strongly incorporate both analysis and design patterns, the outlined methodology allows for a business problem oriented design. Additionally, the reference architecture separates different architectural aspects into different layers.

Since re-use is crucial here, we plan to focus in future work on improving the process of determining appropriate services/legacy functionality.
Additionally, we need to specify a framework for our reference architecture that provides an execution environment for the designed composite applications. This will include the specification of interfaces between the single architectural layers as well as formalizing the access to the data repository. The latter will provide similar mechanisms as the Service Data Objects (SDO) specification [20] proposes with their Data Access Service. Since the framework is not existent yet we can only map some aspects of the refernce architecture to existing solutions. Anyway, in order to gain experience we are currently applying the methodology for itself in a industry case study. After completion of the design we are going to realize the design using the SAP Netweaver [21] product stack.

References

1. Andrews, T., Curbera, F., Hitesh, D., Goland, Y., Klein, J., Leymann, F.: Web service business process execution language for web services version 2.0 - draft. Technical Report 2.0, OASIS (2005)
2. OASIS Open: Web Services Composite Application Framework (WS-CAF) (2003)
3. Feuerlicht, G.: Application of data engineering techniques to design of message structures for web services. In: Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05). (2005)
4. Reijers, H.: A cohesion metric for the definition of activities in a workflow process. In: CaiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Desing (EMMSAD, 03). Velden, Austria. (2003)
5. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
6. Royce, W.W.: Managing the development of large software systems: Concepts and techniques. In: ICSE. (1987) 328–339
7. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
8. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns: Identification, representation and tool support. (2005) 353–368
9. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service interaction patterns. In van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 302–318
10. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. The Addison Wesley Signature Series. Pearson Education Inc. (2004)
11. Papazoglou, M.P., Georgakopoulos, D.: Introduction to special issue on SOC. Commun. ACM **46**(10) (2003) 24–28

12. Decker, G.: Bridging the gap between business processes and existing it functionality. In: Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05). (2005)
13. Dehnert, J., van der Aalst, W.M.P.: Bridging the gap between business models and workflow specifications. Int. J. Cooperative Inf. Syst. **13**(3) (2004) 289–332
14. Wu, B., Lawless, D., Bisbal, J., Richardson, R., Grimson, J., Wade, V., O'Sullivan, D.: The butterfly methodology : A gateway-free approach for migrating legacy information systems. In: ICECCS, IEEE Computer Society (1997) 200–205
15. Linthicum, D.S.: Next Generation Application Integration. Addison-Wesley, Boston, MA USA (2004)
16. Kaufman, G.: Pragmatic ecad data integration. Technical Report 1, New York, NY, USA (1990)
17. Hofmeister, H., Wirtz, G.: A pattern taxonomy for business process integration oriented application integration. In: The 2006 International Conference on Software Engineering and Knowledge Engineering (SEKE'06). (2006)
18. Grefen, P., Vonk, J., Apers, P.: Global transaction support for workflow management systems: from formal specification to practical implementation. The VLDB Journal **10**(4) (2001) 316–333
19. Balushi, T.H.A., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: Performing requirements elicitation activities supported by quality ontologies. In: The 2006 International Conference on Software Engineering and Knowledge Engineering (SEKE'06). (2006)
20. Beatty, J., Blohm, H., Boutard, C., Brodsky, S., Carey, M., Dubray, J.J.: Service Data Objects for Java Specification. Technical report, BEA and SAP and IBM (2005)
21. SAP: (SAP Netweaver)