A Multi-Layered Framework for Pattern-Aided Composite Application Design

Helge Hofmeister
European EAI Department
BASF IT Services n.V.
Brussels, Belgium
Email: helge.hofmeister@basf-it-services.com

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

***Abstract*— In this paper we describe our viewpoint of designing composite applications. We introduce an architectural reference framework that allows for a business process-centred development of composites. The framework groups artefacts of similar abstraction levels as well as concerns at five distinguished layers. The layers also correspond to phases of a design methodology and cover the aspects of composite applications from process-centred orchestration, over transactional coordination to data transformation and connectivity. Based on the framework this paper shows as well how different pattern systems can be used both as design and as analysis patterns in order to facilitate composite applications' design.**

Keywords: SOA, Reference Architecture, Design Patterns

I. Introduction

Service-oriented architecture (SOA) is an emerging spirit that promises to allow for agile software development by re-using existent functionality. The software applications that re-uses functionality are called composite applications.

Inherent aspects of service-orientation 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[1] — 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 standards as [1] that are process-centred orchestration mechanisms for web services. Considering 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. Even if this a promising area and the work already carried-out is of good quality, are these "just" technical standards and ideal thoughts that need to be applied the correct way in order to design automated business processes in a good way.

The quality of a system's design is partially determined by the quality of the basic components' design. Thus, it is important to not only compose services but to compose well designed services. This is why there exist already sophisticated definition processes (such as [2]) or quality metrics (such as [3]) that support the grouping of functionality into services. But despite of the quality 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 boarders, defined by different people and executed by different agents. So it is not applicable to expect well-designed services that are easily to be orchestrated by a business process that is translated into a service orchestration language (eg. using the approach of [4]). We consider the SOA approach and all the benefit of its agility as promising, though. We believe that business requirements that are expressed by business processes should not be constrained by incompatibility of services. Our aim is to provide mechanisms that allow that the requirement engineering is not constrained by the definition of existing services.

This is the reason why we propose an architectural framework that helps to moderate between basic services and business processes. In order to define the framework, we identified several aspects that need to be respected in order to bridge the described gap. Each layer we identified corresponds to phases of a development methodology that might be applied when building composite applications. Additionally, each layer represents both a certain level of abstraction as well as an ideal thought of technical layers a composite application might be build of.

We describe how design patterns fit with the describe architecture. In combination with the hierarchy of layers that is provided by the architecture, design patterns are supportive in terms of system design, effort estimation and model driven development. This is because the set of applicable patterns is limited on each layer and design decisions effect the set of available patterns as well.

After presenting related work in section II, we introduce our architectural framework in section III. The introduction is followed by the discussion of patterns, how they support the actual design and implementation of composite applications (section IV) as well as how design decisions effect the applicability of related patterns. We close with a summary

[1]Even if the transparent distribution is now considered to be cross-organizational without major extra effort.

and an outlook to future work in section V.

II. Related Work

Most important to our work are multiple patterns that have been identified in recent research. These are the workflow patterns presented by van der Aalst et al. [5], the workflow data patterns by Russel at al. [6], the service interaction patterns by Barrows et al. [7] as well as the enterprise integration patterns by Hohpe and Woolf [8]. We analyzed these patterns with regards to their applicability within our framework.

Not in terms of patterns but in terms of service design are the work of Reijers [3] and to Feuerlicht [2] important. The main difference between our work and these service definition methodologies is that our approach is designed to deal with a possibly fixed world with pre-defined application services and business processes.

Multi-layered reference architectures are of course used as well in the area of service-oriented design. Notably is the work of Decker [9] who also describes an intermediate layer between business processes and application services that align processes and IT. Whilst he is solely focusing on the semantic gap between processes and services, we additionally introduce coordination and transaction handling while providing a standardized execution environment as well.

In the area of mapping workflow descriptions to each other Dehnert and van der Aalst did some interesting research [4]. Their approach is to map business process descriptions onto workflow descriptions using petri-nets. This work is complementary to our work as it describes how to derive the fourth level of the presented architecture.

III. Architectural Framework for Composite Applications

According to Linthicum, Business Process Integration Oriented Application Integration (BPIOAI) provides another layer on-top of existent system integration such as Information Oriented Application Integration (IOAI) or Service-Oriented Application Integration (SOAI) (cf. [10]). This means, that system integration-focused technologies such as eg. 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 abilities of 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 integration sub-system.

This chapter presents five layers that together form the complete composite application including the integration sub-system.

A. Layers of the Architecture

The introduced architectural framework provides several layers of abstraction. These abstraction levels loosely correspond to phases of a development methodology. The methodology that is described in [11] implicitly underlies the presented framework.

According to the necessary steps for building composite applications, we identified five layers that composite applications should be build of. These layers are hierarchical in the way that lower layers provide functionality to the upper layers. An image showing all layers can be found in figure 1.

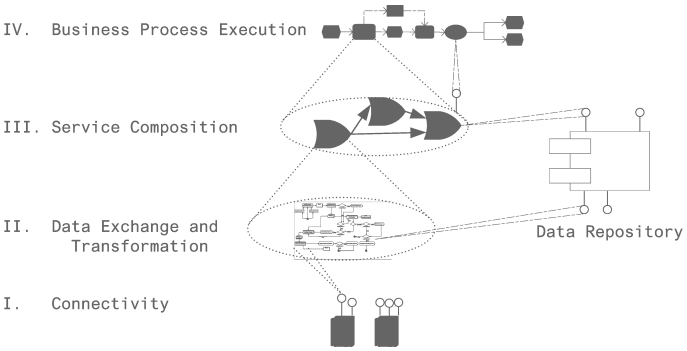

Fig. 1. Architectural Layers

1) Layer Zero - Legacy Application Systems: BPIOAI and composite applications are more agile and flexible orchestrations of existent functionality. This functionality is usually provided by the application systems of an organization — by the means of *application services* that are exposed by these services in some way.

There exist some proposals that aim at supporting the definition of application services. However, application services are designed in order to allow re-assembling existent functionality of application systems in composite applications. This is why their definition is dependent on the possibilities of the application system providing their functionality. Anyway, designing a service always has the requirement of designing a re-usable service. Because of this, there is a broad variety of how granular or cohesive these services are. Due to that reason we consider it as unrealistic to have the ability to work with services that are completely aligned with the needs of business processes. This is why our framework aims at supporting any *size* of application 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. [10, p. 218]). How application services are technically connected to service consumers, is described at the subsequent layer.

2) Layer One - Connectivity: In order to allow the usage of application systems' 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. This layer provides connectivity in between the legacy application system and the composite application by homogenizing the protocol that is used to access functionality. From the composite application point of view, this layer provides the application services.

The characteristics of transparent distribution of services is

achieved by the connectivity layer since it converts a common protocol all basic services are dealing with into the specific protocol of the connected legacy system. Usually this layer's functionality is provided by adapters. Basically, adapters "[...] could be a set of 'libraries' that map the difference between two distinct interfaces [...] and hide the complexity of those interfaces [...] from the developer" [10, p. 218]. Thus, at this connectivity layer the heterogeneous legacy systems are encapsulated by an interface as it is provided by an adapter. There exist already several standards for building adapters or providing a common protocol application systems might support. Two examples of standards we consider as being located at this layer are the Java Connector Architecture [12] and the web service standards WSDL [13] in conjunction with SOAP [14]. Since there are existent standards, we do not describe this layer more in depth and refer to the according standards and vendors of adapters.

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) Layer Two - Data Exchange and Data Transformation: This layer of data exchange and data transformation provides the functionality to the upper layers that is usually provided by an integration server in Enterprise Application Integration (EAI) scenarios. Thus, this layer describes the integration subsystem of the composite application.

This layer is dedicated to tackling down 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 increased. 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. [8, pp. 355-360] or [15]). After unification of the data, the data is stored into a data repository we introduce as part of the framework 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.

As the business and control data is stored into that repository, upper layers can merely deal with a standardized ticket that reference the data set in the context. So the interfaces of the involved upper services can be defined independently of the actual data formats (even independently of the canonical data format).

Besides the data transformation, this layer also provides functionality 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[2]. The benefit is that upper layers do not need to deal with this sort of errors and can therefore provide more re-usable implementation of business logic.

Additionally, this layer 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.

Technical routing is performed at this layer as well. This is being performed by dedicated services that perform the connection with service directories.

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 and integration flows you might consider [16].

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.

Without further concepts, the functionality that is provided by these services would be determined by the functionality offered by the application systems. In order to apply the paradigm of service orientation and compose new functionality out of existent or to enrich functions within a specific context, it might be appropriate to combine the application specific functionality to new functionality. Expressed differently, application services might be needed to be aggregated to more problem-oriented services (*enterprise services*). This is why we introduce this layer that leads the invocation of two to n basic services with a flow in order to form the enterprise services.

Conceptually, the 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. The benefit of this approach is that the orchestration themselves can remain flexible and their re-usability is increased for that reason.

This layer's coordination might be appropriate due to different requirements. Considering business processes and functionalities that are needed at the level of business requirements, the basic single services need to interact in a certain way in order to meet the business requirements. Thanks to the homogenization means provided by the data exchange and transformation layer, these aspects are transparent to this layer. As a consequence, the coordination layer can be used to aggregate services with strong focus on functional behaviour. The defined functional flow might aggregate both, company internal or external services to services that are in turn usable both internally and externally which introduced the need to support business protocols, too. These business protocols are

[2]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).

sets of actions that have to be performed by multiple parties in order to allow successful execution of certain business functionality (cf. eg. [17]). A technically motivated aggregation, such as the realization of multi-resource protocols that ensure consistent state transitions, might be appropriate at this layer, too.

These consistent state transitions can be ensured by two means of transaction handling. Rather short-term transactions fulfilling the ACID properties by locking and rollback mechanisms or more long-term transactions without locking and compensation actions. Of course are the orchestrated functionality implemented at the application systems and the consistent state transition is to be assured by these systems.

Orchestrating the services of these systems does, however, raise the need for a cross-service transactional coordination as well. Even if the application services have to support the transactional coordination by appropriate compensation operations or supporting transactional interfaces, the coordination itself has to be controlled at this layer.

According to Grefen, the transactional coordination consists of two layers [18]. One layer for so-called *local transactions* with ACID properties and one for *global transactions* with relaxed transactional properties.

In our approach local transactions are 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, Grefen et al. relax the isolation and atomicity properties and introduce the mechanism of safepoints. [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.

Summarizing this layer of service coordination, we propose one layer that transparently coordinates application services to — what we call — enterprise services. These enterprise services basically retain the generic interface of the inbound and outbound services as they are exposed by the second layer

5) Layer Four - Business Processes: As we stated in the introduction, our aim is to provide means for dealing with complexity so that business processes can be executed by a machine very easily. This process execution layer is this 5th layer. In order to be executed, the business process description has to be transformed into a workflow execution language that is compatible with the common protocol used at the other layers. An example for the web service world would be the business process execution language for web services (BPEL4WS). Such a description orchestrates the services that are exposed by lower layers and interacts with human users for data input/output or support of decisions. Fundamentals of this transformations are described by Dehnert and van der Aalst [4].

Human-interaction is realized again with services that are called from the user interface and enterprise services that are called by the user interface. In this area performance plays an important role because user are directly affected by any delay. Also are interface technologies and process execution layer often bundled into single products. We treat user interactions in our model as interactions with legacy applications[3]. As we do not provide statements for the intra application system design, we neither provide guidance to the design of the user interaction.

Process branching based on certain indicators are required in order to provide an execution environment for workflow descriptions. This is why 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 being 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.

IV. Hierarchical Pattern System for Composite Applications

In the description of our implicit development methodology, we stated the artefacts from some phase might be used as input for another phase in order to support design decisions in this subsequent phase. In this section we discuss design patterns for the single layers and show interconnections between patterns of different layers.

A. Intra-Layer Patterns

1) Patterns for the Layer of Business Processes: Business processes are usually described by workflows in an imperative way. Workflows have several aspects or perspectives that together form the description of a workflow. These perspectives are the control-flow, data, resource and operational perspective (cf. [5]).

The control-flow perspective "describes activities and their execution ordering through different constructors, which permit flow of execution control, e.g. sequence, choice, parallelism and join synchronization" [5, p. 2]. Business and processing data that describes data exchange between a workflow's tasks and pre- and post-conditions for the tasks are described in the data perspective. The resources and the operational perspective describe how workflows are executed in terms of organization respectively application systems.

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 the layer of business processes. This is because the sequence of underlying tasks that are represented by services is determined at this layer. Thus, it is important to have means in place that standardize the description of the sequence as well as how decisions (e.g for forking and merging) are supported.

[3]Of course, user interaction could be supported by legacy application systems.

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. [6] 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 artefacts that can be identified within a workflow description[4]. 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 ibidiem.

2) Patterns for the Layer of Service Coordination: The aspects of composite applications that deal with multilateral service coordination are described at this layer. So are design artefacts that provide standardized means for describing and analyzing coordinations. As the service coordination layer serves multiple purposes, we show the related design artefacts also on a per purpose basis.
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 reference the service interaction patterns by Barrows et al. [7]. The authors distinguish their patterns into four groups: Single-transmission bilateral interaction patterns, single-transmission multilateral interaction patterns, multi-transmission interaction patterns and routing patterns. Three basic dimensions form these groups. The first dimensions indicates the maximum number of services involved in an interaction. The maximum number of exchanges between two parties involved in a given interaction are described in the second dimension (single transmission or multi-transmission). In the case of two-way interactions it can be distinguished whether responses are sent back to the requester or to a third service. This is described by the third dimension. All of these patterns can possibly be identified at this layer. Not for the sake of analyzing but for designing the workflows at this layer for the purpose of assembling application services in a way that meats the business process description, we reference five design patterns. Decker introduces a so-called process support layer that deals with various aspects of incompatibility between business process tasks and application services [9]. As we provide a more granular framework, the process support layer cannot be mapped directly to our layer of service coordination. Anyway, most of the patterns of two important classes of patterns are relevant: The patterns for granularity problems as well as the patterns for interdependency problems.

[4]Of course, they are also useful for designing engine-specific workflow descriptions out of business process descriptions.

At our layer of service coordination the following patterns are considered helpful: *Composition*, *Decomposition* and *Bulk Service* for the class of granularity problems as well as the *Sequentializing* and the *Reordering* patterns for interdependency problems. All these patterns deal with the interaction among more than two entities and are to be used according to the business process model as well as to the existent application services.

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].

1) **Local Transaction Composition**. *Definition:* A Local Transaction Composition (LTC) is a subgraph of a given service orchestration that is required to be atomic, consistent, isolated and durable. Atomic means that the context of the composition is after its completion either in the initial state or in a consistent end-state. Isolated means that the context is not accessible during the composition's execution. Durable means that the composed application services change their state in a way that subsequent read-operations represent always the new state. The LTC forms itself a stateless service.
2) **Global Transaction Composition.** *Definition:* A Global Transaction Composition (GTC) is a subgraph of a given service orchestration that needs to be consistent, durable and compensable. A GTC can be an orchestration of both atomic services and other orchestrations and forms itself a stateful service with multiple operations.

In order to support these two transactional patterns, we require three properties that services or transactional compositions might have. These are: Safepoint, Idempotent and Compensation. Not 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.
GTCs can be described by data visibility, data interaction and data transfer patterns that are described in [6]. 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.
All patterns that are relevant to this layer are not only input for lower layers' designs. Also the identification of certain patterns affect the design within the coordination layer.

Example: In order to demonstrate the usage of patterns and some interdependencies, we provide a small example.
An enterprise service *SendInvoice* that creates a new invoice

and returns the total turnover of a customer is required by a certain business process. While there are four single application services the business process is designed with one task and therefore only one service call is implemented in the corresponding fourth level workflow. In order to compose these services to one enterprise service, the *Composition* pattern needs to be applied. As a design decision that needs to be made when applying the *Composition* pattern, it is described that the first three services need to be invoked all together or no at all. So from the need of a *Composition* arises the need for a LTC that spans the services *CheckRating*, *CreateInvoice* and *SendInvoice*. The context of this LTC is only accessible during its execution by these three services. So the workflow data pattern two — *Block Data* — needs to be applied. In order to provide the customer number and intermediate results (such as the customer's rating) to subsequent services, the data pattern 9 — *Block Task to Sub-Workflow Decomposition* — needs to be applied. As the customer number is also required to execute the *GetTotalTurnover* service, the customer number needs to be transferred as a parameter between the LOC-block and the single application service and the data pattern 8 — *Task to Task* — needs to be applied. As soon as the LOC-block is finished, its context needs to be committed and to be made available to all services of the process. Thus, the data pattern 14 — *Task to Environment - Push Oriented* — is required in order to commit the local context to the global one.

3) Patterns for the Layer of Data Exchange and Data Transformation: The patterns we assign for this layer are the Enterprise Integration Patterns by Hohpe and Woolf [8]. In order to structure these patterns we introduced integration flows that form a taxonomy for the integration patterns [16]. This taxonomy describes standard services that are required to read or write data to application services, respectively. In addition to the functionality provided by the integration patterns, the integration flows additionally deal with data heterogeneity and communication semantics. This why optional constructs are incorporated into the generic integration flows. Such constructs are for instance optional acknowledgments that are triggered in certain states of the integration flow in order to support certain communication semantics. This optional functionality is considered as design pattern as well. All relevant patterns and the categorizing taxonomy are described in [16].

V. Summary and Outlook

We have presented our an architecture for composite applications. The benefit of our proposal is that it allows for a business problem oriented development methodology as it separates different architectural aspects into different layers. With well-known patterns we provided support both for analyzing as well for designing the architectural layers. This leverages a design methodology and is a first step towards a semi-automated development approach of composite applications. Altogether, the architecture, patterns and pattern dependency form a multi-layered framework for pattern-aided composite application design.

In order to validate the presented concepts we have identified a use case from the context of an IT service provider. We used the presented architecture and patterns in order to design a composite application supporting the identified business process. Currently we are busy to finish the implementation of this industry-scale use case. For the implementation we used a mapping of the described architecture onto a target platform in order to facilitate the implementation as well. Intermediate results show that the described architecture also supports the proper application of third-party platforms.

References

[1] T. Andrews, F. Curbera, D. Hitesh, Y. Goland, J. Klein, and F. Leymann, "Web service business process execution language for web services version 2.0," OASIS, Tech. Rep. 2.0, December 21 2005.

[2] G. Feuerlicht, "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. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0FD9B681AADEFFA1852570CF005FDC06/$File/RC23819.pdf

[3] H. Reijers, "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.

[4] J. Dehnert and W. M. P. van der Aalst, "Bridging the gap between business models and workflow specifications." *Int. J. Cooperative Inf. Syst.*, vol. 13, no. 3, pp. 289–332, 2004.

[5] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[6] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst, "Workflow data patterns: Identification, representation and tool support." pp. 353–368, 2005.

[7] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service interaction patterns." in *Business Process Management*, W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, Eds., vol. 3649, 2005, pp. 302–318.

[8] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*, ser. The Addison Wesley Signature Series. Pearson Education Inc., 2004.

[9] G. Decker, "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. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0FD9B681AADEFFA1852570CF005FDC06/$File/RC23819.pdf

[10] D. S. Linthicum, *Next Generation Application Integration*. Boston, MA USA: Addison-Wesley, 2004.

[11] H. Hofmeister and G. Wirtz, "Approaching a methodology for designing composite applications integrating legacy applications using an architectural framework," in *Proc. GI-FG Treffen EMISA, Hamburg, LNI, Vol. 95, Springer*, 2006.

[12] *JSR 112: J2EE Connector Architecture 1.5*, Sun Microsystems Inc. [Online]. Available: http://www.jcp.org/en/jsr/detail?id=112

[13] *Web Services Description Language (WSDL) 1.1*, W3C, March 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[14] *SOAP Version 1.2 Part 1: Messaging Framework*, W3C, June 2003. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[15] G. Kaufman, "Pragmatic ecad data integration," New York, NY, USA, Tech. Rep. 1, 1990.

[16] H. Hofmeister and G. Wirtz, "A Pattern Taxonomy for Business Process Integration Oriented Application Integration," in *Proc. 18th Intern. Conf. on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2006*, 2006.

[17] "Rosettanet partner interface processes," RosettaNet, Tech. Rep. [Online]. Available: http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity[OID[9A6EEA233C5CD411843C00C04F689339]]

[18] P. Grefen, J. Vonk, and P. Apers, "Global transaction support for workflow management systems: from formal specification to practical implementation," *The VLDB Journal*, vol. 10, no. 4, pp. 316–333, 2001.