A Template-driven Approach for Maintainable Service-oriented Information Systems Integration

CLAUS PAHL and YAOLING ZHU and VERONICA GACITUA
Dublin City University, School of Computing
Dublin 9, Ireland
[*Claus.Pahl*|*Yaoling.Zhu*|*Veronica.Gacitua*]*@computing.dcu.ie*

Abstract. Service-oriented architecture (SOA) is currently the predominant software integration framework. Web services provide the predominant platform for SOA. SOA as an integration architecture solution supports a range of application scenarios. We present a solution for the integration of business information systems based on SOA and Web services. We discuss maintainability requirements in relation to information-specific integration and architecture aspects. A template-based approach based on modular and declarative transformation rules and architectural styles and patterns defines our solution to the maintainability problem of integration architectures.

Keywords. Service-oriented Architecture, Enterprise Application Integration, Service Architecture Description, Information Integration, Integration Architecture.

1. Introduction

Service-oriented architecture (SOA) [13] is rapidly gaining momentum as a software architecture and platform technology that allows the integration of enterprise-wide system architectures. Enterprise architecture integration (EAI) [5], which is concerned with the integration of legacy systems, off-the-shelf components and new applications, has been provided with a new level of interoperability through SOA and Web services [1].

Information integration is an aspect that is often neglected in current research into SOA and service-based EAI approaches. Service integration cannot be achieved without information integration for heterogeneous applications. Information integration is a common problem for EAI in general [10]. The business model of application service providers (ASP) is an archetypical example of business information systems that relies on the integration of different data representations between provider and client [27]. It clarifies the need to reconcile heterogeneous information architectures within a service-based application system architecture. While information integration is necessary to reconcile data heterogeneity across different systems, applications, organisations and platforms, the current integration solutions are often ad-hoc and

limited in terms of tool support. Often, for instance XML-based procedural XSLT transformations are used for data integration.

Equally, on the service integration and composition level, service orchestrations that integration services of a possibly distributed and heterogeneous application are implemented without a proper architectural design. The executable orchestration languages, such as WS-BPEL, that are used do not provide sufficient architectural support for application and information integration.

The information and software architecture of an application system is crucial if quality is taken into consideration. Maintainability is a software quality that needs to be guaranteed if a system is subject to change and evolution. Maintainability, however, is dependent on adequately designed individual applications. These observations apply in particular, if the integration of different individual systems is under consideration.

We aim here to provide a maintainable solution for service-oriented information systems integration in the SOA and EAI context. Our contribution is an approach to integrate and embed information integration into service-based application integration. Although based on different techniques, service-level and information-level integration are equally important. We investigate these aspects under consideration of maintainability as the quality goal.

- We provide an information-driven service architecture approach for application integration. Services deal with information aggregation and information processing in a 4-layer architecture. Information drives service design and composition.
- The integration architecture is maintainable through modular, declarative information transformation and abstract architectural patterns and integration styles. We use these two types of templates, i.e. rules and styles/patterns, as structuring mechanisms for a maintainable integration architecture.

We aim to move the current focus on runtime architecture and platform towards software architecture and service engineering.

The starting points of our solution to a maintainable integration architecture are individual service and data models or schemas that define each application to be integrated. Application integration in general is based on the definition of information architectures from domain models and the definition of service architectures from business process models. This is based on a dual service and information refinement approach. Information integration incorporates information aggregation (where several source provider services provide input to an integration service) and information processing (where processing services are orchestrated) as the core function. Service architecture is concerned with the composition of services to orchestrated service processes.

Section 2 provides some background on integration problems and discusses related work. In Section 3, the integration problem and its solution are outlined and we introduce a modelling notation that forms the core of our approach. Our technical contribution, a maintainable integration approach through templates, is presented in Section 4. We evaluate the solution from the maintainability perspective in Section 5. We end with some conclusions.

2. Integration Architecture Context and Related Work

IT architecture and application integration have received much attention recently with the emergence of SOA and the Web Service platform [1,28]. The services computation platform provides a flow-based composition and integration model for software services [29]. Information integration activities in SOA, however, need to be considered in these solutions. We look here at business information systems and discuss their IT architectures and application integration problems.

The state-of-the-art of application integration can be described as follows. Information integration is often based on XSLT as a procedural transformation language, which is directly based on XML [12]. Service integration is often achieved using WS-BPEL as a service orchestration language [34], which is typically used ad-hoc. Although XML enables interoperability or data representations and WSDL and WS-BPEL do this for service description and invocation, the languages are low-level with no suitable abstraction mechanisms provided. The nature of these commonly used languages gives rise to some limitations [35]. Maintainability is a problem. Static and behavioural structures are difficult to recognise due to a lack of abstraction. Specifications are difficult to read and write. Both XSLT and WS-BPEL are XML based, which is not meant as a design language. XSLT provides little structure. The languages are applied ad-hoc if no methodological development support is provided. Scalability is limited. Only abstraction would allow more complex systems to be built. Other aspects such as performance can also be negatively affected, but these concerns shall not be addressed here.

In [24], a business rule engine-based approach is introduced to separate the business logic from the executable WS-BPEL process to improve design and maintenance through abstraction. A declarative, rules-based approach can be applied to the data transformation problem [18,22]. The difficulty lies in embedding a declarative transformation approach into a service-based architecture in which clients, mediators, and data provider services are composed. Zhu et.al. [37] and Widom [31] argue that in particular the heterogeneity of data formats in service-based environments make mediated architectures more suitable than data warehouses or federated schema systems. Semantics can play an essential role in these architectures. Haller et.al. [9] propose semantic service mediation to achieve consistency between the mediated services. Data and service abstraction is in principle a contributor to maintainability, but additional measures can further enhance the situation, as we will demonstrate. We propose a solution where information integrates services, i.e. an information-driven approach to service integration.

3. Integration Architecture Modelling

Integration architecture is the methodological framework within which we embed our information and service integration techniques. We define integration architecture as a development process aiming at an integration of application architectures consisting of, firstly, a global information architecture based on a semantic information model and, secondly, an integrated service architecture based composed and orchestrated service processes. This section outlines the techniques and methods on which we build our approach and it introduces our modelling notation.

3.1 IT Architecture Layers and Stages

Integration architecture is an aspect of the wider scope of IT architecture [15]. A number of IT architecture models influence and constrain our approach.

- At the business level, the business domain model defines the domain organisation model and the business processes. These exist for each organisation and govern existing applications in terms of both data and service aspects.
- At the application level, individual applications (existing or planned) are described in terms of local data models (in the form of schemas or taxonomies) and service descriptions (in abstract service description languages like WSDL or other interface definition languages).

Between the business domain model layer and individual application layer, we place an integration architecture layer (IAM) that provides the location for our techniques. We propose a staged development approach based on these layers. At the business domain model (BDM) layer, the focus is on business and domain aspects. This layer acts as a provider of process and information models for the actual application integration. At the integration architecture model (IAM) layer, the focus is on application systems in terms of their boundaries and interactions in order to model and implement their integration.

3.2 Service Architecture Modelling

The approach to application integration should focus on architectural concerns. The objective of software architecture [2] is the separation of computation and communication. Architectures are about components (i.e. loci of computation) and connectors (i.e. loci of communication). This allows a developer to focus on structures and the dynamics between components separately from component implementation. Various architecture description languages (ADL) and modelling and development techniques have been proposed [16]. An architectural model captures common concepts found in a variety of architectural description languages:

- components provide computation,
- interfaces provide access to black-box components, and
- connectors provide connections between components.

In service architecture, the main emphasis is on the composition of services to processes and on the overall configuration of services and service processes.

We enhance this architecture view by a semantic information architecture. Ontologies are knowledge representation frameworks formalised in an ontology language (such as OWL) [6,30], which is usually based on a terminological logic (such as description logic). Knowledge is represented in form of concepts and (quantified) relationships between these concepts to characterise them semantically.

Modelling of architecture constraints is the central development activity here. We propose a layered architecture development approach. We introduce a modelling language consisting of two notations that supports the two focal aspects of information and service integration:

- a service-centric composition and process notation as the service architecture language,
- an ontology-based data schema notation as the information architecture language.

The proposed modelling language acts as an architecture description language (ADL). This service-centric ADL is the central element of a two-layered information and service modelling

technique for the BDM layer and also the logical, platform-independent aspects of the IAM layer. This service-centric architecture modelling language is characterised as follows: services are the basic building blocks and service processes and service interaction are the central modelling aspects. We define our language using service modelling concepts from the Business Process Modelling Language (BPMN) [17,19] and information modelling concepts from ontology languages [6].

3.3 Information Modelling

Architecture as an activity is about modelling, where an integrated and coherent notation is desirable. Business domain modelling (BDM) consists of a combination of process (business process) and information (organisation, product) modelling. Based on a core service process modelling notation, we propose two enhancements:

- Firstly, an information architecture shall be added in the form of a taxonomy. The first step results in a service model with associated information architecture, as the Figure 1 illustrates. It adds a taxonomy of information model terms (called concepts). These are hierarchically structured and can cover the organisational or product dimension. Figure 1 illustrates the inclusion of the central process activity into a domain organisation hierarchy for a banking application, directly associated to a process model.
- Secondly, the taxonomy is enhanced to a semantic information architecture in the form of an ontology, which is presented separate from the process model [23]. The second step enhances this information taxonomy to a semantic information architecture. It essentially enriches the taxonomy towards a richer ontological model with data/object properties, from which a canonical XML representation can be generated. Figure 2 captures a Customer data structure as an ontology based on concepts (customer, service) and the properties that connect the concepts (data-valued properties such as supportID and object-valued properties such as usedServices).

Both the withdrawal process and the customer object are domain entities modelled at the business model layer. Both are concepts that can be represented as part of the Front Office concept of the domain information taxonomy.

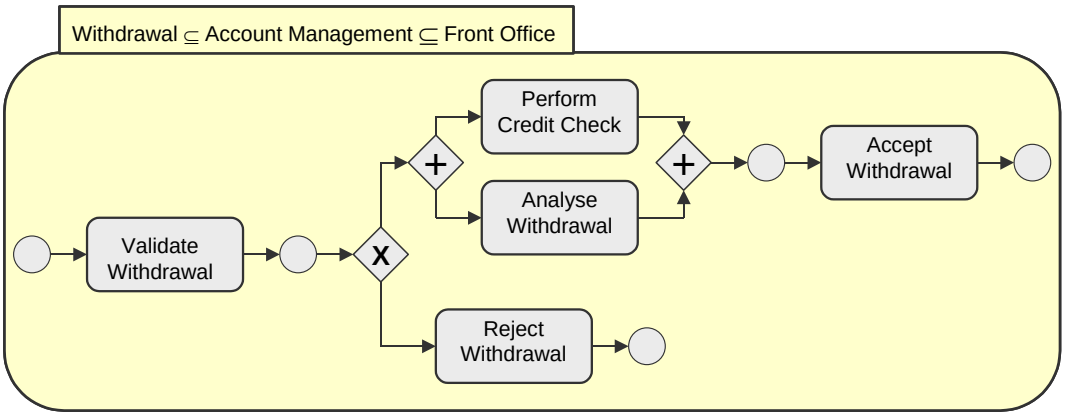

Figure 1. Information taxonomy-enhanced service process.

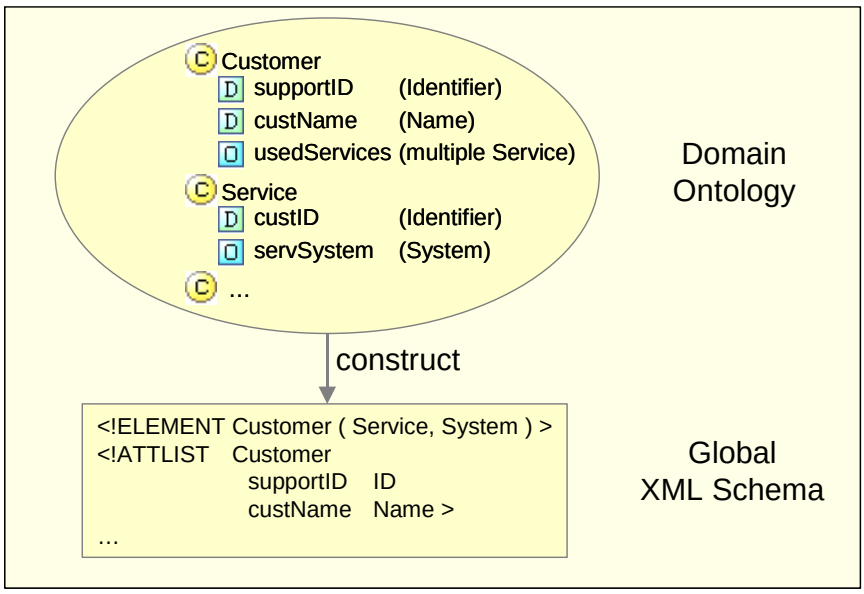

Figure 2. Ontology-based information architecture.

4. Integration Architecture – Maintainability and Templates

4.1 Integration Principles

The integration of software application systems on the Web platform level is our objective. This platform is characterised by XML as the data representation notation and Web services as its computational abstraction. The wide application of the Web platform guarantees interoperability. We distinguish information and service integration based on the two interoperable Web technologies. The principles of information-driven integration modelling shall be introduced in this section.

- Information integration is based on data transformation. Schemas (local schemas for individual applications and global schemas for application integration) provide the core representation. Transformation languages (procedural or declarative) define mappings between local and global representations.
- Service integration addresses the composition through orchestration and choreography. Abstract interface descriptions of services (in the form of WSDL) describe individual services. Collaboration specifications (in the form of WS-BPEL or WS-CDL) define service compositions.

4.2 Template-based Information-driven Integration Architecture

We propose an information-driven integration approach for service-based application integration. Services provide the computational abstractions. However, these services can only interact if data representations are integrated. Specific service functions can deal with information integration aspects. Information aggregation and information processing functions are provided as integration platform services. These are the two lower layers of a widely used 4-layer

architecture consisting of data, function, process, and presentation layers from bottom to top [11]. A service categorisation – based on the idea of services being information processors or functions – helps to structure the overall application architecture. A layered, information-oriented service categorisation for business information systems organises functionality according to architectural aspects. Service composition to support complex information processing can be achieved through an information flow-based integration model for service composition.

The information-driven service integration support is based on two separate architecture layers:

- The basic layer of information integration (defined through declarative and modular transformation rules) implements information integration through mediation between data sources and users.
- On top of the information integration layer is a service integration layer (defined through architectural styles and patterns) that constrains service integration and embeds information integration.

The development and modelling is therefore service-based, but information-driven, see Fig. 3.

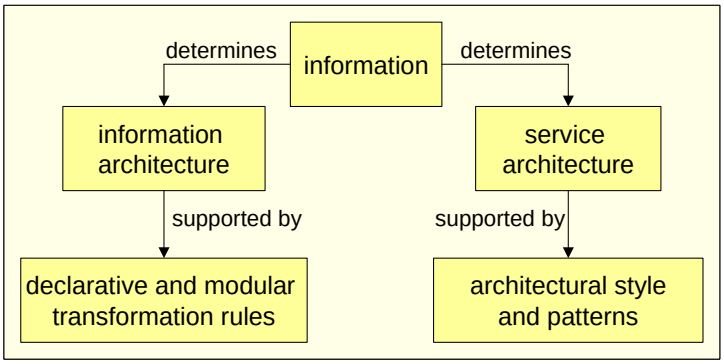

Figure 3. Central components of the integration architecture solution.

Our proposal is to provide more abstraction through the use of templates in order to address the maintainability objective – guided by the following ideas and requirements:

- For information architecture aspects, abstract transformation rules that are easy to maintain through adherence to principles of declarativity and modularity.
- For service architecture aspects, architectural styles and patterns (of interaction behaviour between services) introduce explicit and reusable architectural constraints at the type (meta-model) level of architecture descriptions.

Both are based on the information architecture. We achieve increased maintainability through architectural separation of information integration (transformation) and service integration (composition and orchestration).

4.3 Information Integration Templates: Transformation Rules

A key requirement to overcome the limitations of XSLT and similar languages is a declarative transformation specification language [25]. The query and transformation language Xcerpt [4,26] is designed for querying and transforming standard Web data (XML, HTML) and Semantic Web

data (RDF, OWL). Based on a comprehensive analysis of transformation languages, Xcerpt has emerged as the most suitable one to be used to supported a mediate service integration architecture [36]. Its design principles include:

- declarative transformation rules similar to declarative database query languages,
- separation of matching and construction part to distinguish the concerns of information filtering and information structuring,
- goal-based query programs and transformation rules are distinguished to enable modular rule definitions,

Xcerpt is based on a pattern matching approach to information querying and transformation that allows for a high degree of abstraction and problem-oriented specification. A layered approach of transformation specification through rules achieves compositionality of rules [36]:

- Ground rules are responsible for populating XML data in form of Xcerpt data terms – these are tightly coupled to data provider services.
- Intermediate composite rules consume the Xcerpt data terms – these integrate ground rules to render global schema data types.
- Goal-level composite rules provide data objects for the mediator services based on customer requests.

Rule 1: This rule produces the CustomerArray by grouping and reconstructing.

```
CONSTRUCT
   CustomerArray [[
      all var customer,
      all var supportidentifier,
      all var services [[
         var customerName,
         all var system [[ var systemId,  all var machine ]]
      ]]
   ]]
FROM
   Customer [[ var customer, var supportidentifier ]]
AND
   Service [[var services [[ var system [[ var machine]] ]] ]]
```

Rule 2a: This rule gets Customer data terms according to the global data model.

```
CONSTRUCT
   Customer [[ var customer, all var supportidentifier ]]
FROM
   arrayOfCustomer [[ var customer, var supportidentifier ]]
```

Rule 2b: This rule gets Service data terms according to the global data model.

```
CONSTRUCT
   Service [[ var service [[ var system [[ var machine]] ]] ]]
FROM
   arrayOfService [[
      var service [[ var system[[ var systemId ]] ]]
   ]]
AND
   Machine [[ var machine, var systemId ]];
```

Rule 3: This construct rule gets Machine data terms.

```
CONSTRUCT
   Machines [[
      all machine-of-system [[ var machine ]],
      var systemId
   ]]
FROM
   machineItem [[ var machine, var systemId ]]
```

Figure 4. A layered Xcerpt specification of four modular transformation rules.

Fig. 4 illustrates four modular Xcerpt transformation rules – each based on a matching (FROM) and a construction (CONSTRUCT) part – that allow the transformation of a complex customer information schema. Each rule addresses a specific customer substructure, such as CustomerArray, Customer, Service, or Machine, which makes each rule focussed and easy to understand and to maintain.

In terms of the architecture, building blocks called connectors implement the information-level integration. A connector that translates between data source and customer query is defined through a top-level transformation goal. Backward goal-based rule chaining is the technique that

is applied to compose individual, modular rules for a specific integration request [14]. Variable bindings of constituent rules are chained to the query program itself, which means that data is constructed bottom-up through recursive rule application. For instance, the four rules from Fig. 4 can be composed in this way, starting with Rule 1 as the goal, to create a connector that transforms a complex customer array data structure.

Maintainability of information integration, which is our main objective, is achieved through

- the declarativity of the language itself, which specifies transformations at more abstract, problem-oriented levels,
- the Xcerpt structuring mechanisms that make transformations easier to write and read than procedural languages,
- the proposed layered, modular definition of composable rules that each target specific data constructs,

The semantic enhancement through an ontology-based information architecture provides additionally consistency and consequently simplifies maintenance [21]. In terms of an overarching ontology for an application domain, such as the customer definition in Fig. 2, local data schemas on which the transformation rules are based can be mapped onto the ontology. These mappings can be used to validate the consistency of the transformation, i.e. that semantically equivalent concepts are mapped onto each other.

4.4 Mediated Integration Architecture

A link needs to be created between the information integration ideas just presented and the service integration description. A mediated information and service integration architecture is the solution [7]. Mediation provides an infrastructure that allows providers and users to be connected and heterogeneous information and services to be made compatible. We use layered integration engines for the different aspects of integration. Our layered mediation architecture is presented in Fig. 5.

Three engines on different layers implement the mediation functionality [32] of the architecture, whereby the first two engines provide support for information integration at the lower layer and the third supports service integration at the higher layer.

- The information integration engine executes the data transformations. It uses a rule repository to store modular Xcerpt transformation rules. Depending on a client request, the connector generator assembles a connector (a composite, goal-oriented transformation) from the individual modular rules. The Xcerpt engine then executes the transformation based on the requested input data.
- The mediation engine connects data providers and users through a mediation workflow, i.e. it mediates between user queries and data providers. It uses the integration engine, which is a separate and independent service, to carry out the necessary transformations.
- The application process engine implements an abstract application process as an orchestrated process based on individual application services. In this process, it includes mediator engine invocations. The application process engine embeds information integration into the service layer by acting as the user for the mediator engine.

Our approach is a 2-layered integration with a service integration layer and an information integration layer – the latter consisting of mediation and integration engine.

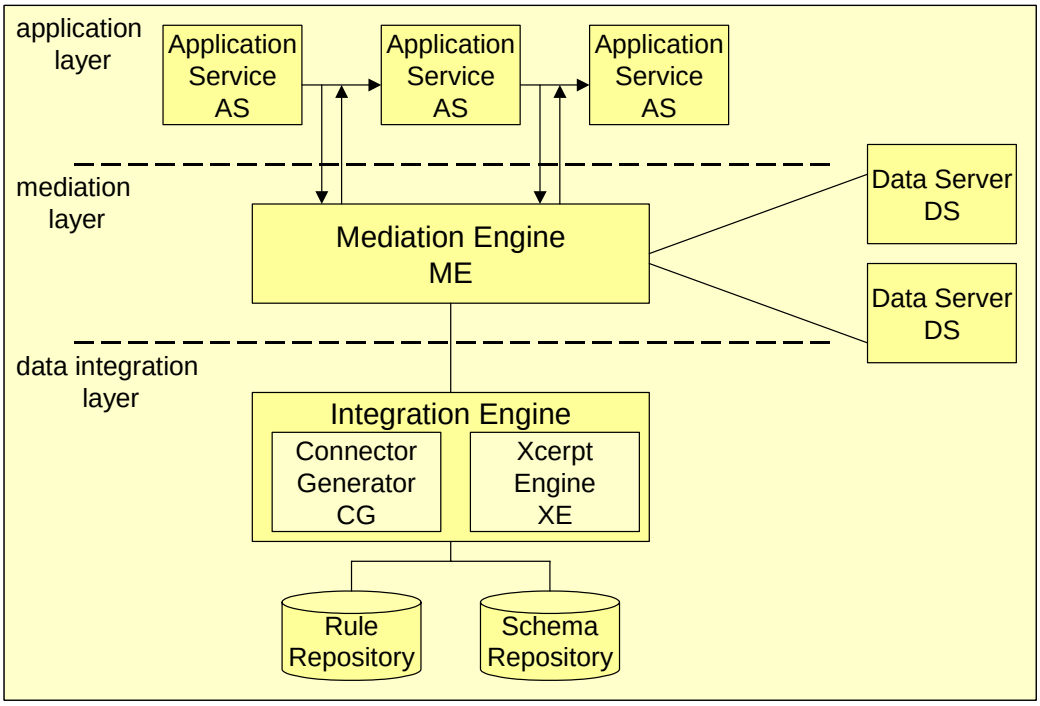

Figure 5. Layered integration architecture.

The proposed layered architecture provides maintainability benefits. Separation of information and service integration aspects is achieved, which means that the impact of change is localised. Within the information integration part, Xcerpt specifics are encapsulated and decoupled from the mediation layer, i.e. other languages for transformation can replace Xcerpt without any impact on the mediation and service integration processes as such at the higher layers.

4.5 Service Integration Templates: Styles and Patterns

The information architecture and information integration solution determines the service integration solution. We suggest a specific architecture style to support information processing and integration for SOA. This integration style is based on the previously presented mediator architecture with application process services:

- The style itself is formulated as a type-level constraint for an architecture definition for an application system. The architecture and its constraints can be formulated in a common architecture description language (ADL) such as ACME [8], see Fig.6.
- The style captures the presented three layers and the mediator and integration engines, which are provided as services in this architecture. The integration style captures and constrains the lower two layers of the 4-layer service architecture.

The style definition formalises the structural constraints of the integration architecture illustrated in Figure 5. An architectural style shall here formulate an architecture template that can be instantiated and refined in a concrete integration project. It is therefore formulated at the type level, i.e. it predefines architecture elements and their properties.

```
Family IntegrationArchitecture = {
    Component Type AS = {
        Ports {In,Out} };
    Component Type ME = {
        Ports {AS-In,AS-Out,DS-In,DS-Out,CG-In,CG-Out} };
    Component Type DS = {
        Ports {In,Out} };
    Component Type CG = {
        Ports {ME-In,ME-Out,XE-In,XE-Out} };
    Component Type XE = {
       Ports {In,Out} };


    Connector Type AS-ME = {
        Roles {requestInt,provideInt} };
    Connector Type ME-DS = {
        Roles {requestData,provideData} };
    Connector Type ME-CG = {
        Roles {requestTrans,provideTrans};
        Property {count instances(ME-DS) <= 1} };
    Connector Type CG-XE = {
        Roles {requestExec,provideExec};
        Property {count instances(CG-XE) <= 1} };

    Attachments = {
        AS.Out to AS-ME.requestInt;      ME.AS-In to AS-ME.requestInt;
        ME.AS-Out to AS-ME.provideInt;   AS.In to AS-ME.provideInt;
        ME.DS-Out to ME-DS.requestData;  DS.In to ME-DS.requestData;
        DS.Out to ME-DS.provideData;     ME.DS-In to ME-DS.provideData;
        ME.CG-Out to ME-CG.requestTrans; CG.ME-In to ME-CG.requestTrans;
        CG.ME-Out to ME-CG.provideTrans; ME.CG-In to ME-CG.provideTrans;
        CG.XE-Out to CG-XE.requestExec;  XE.In to CG-XE.requestExec;
        XE.Out to CG-XE.provideExec;     ME.XE-In to CG-XE.provideExec;
    };

    Property {
        AS ∪ ME ∪ DS ∪ CG ∪ XE = IntegrationArchitecture ;
        AS ∩ ME ∩ DS ∩ CG ∩ XE = ∅
    }
}
```

Figure 6. ACME-based integration architecture style definition.

Most ADLs, such as ACME, support the component-and-connector view [8]. In terms of this view, the style defines the following types of components:

- The Application Services (AS) provide functionality that contributes to the overall application function of the system.
- The Mediation Engine (ME) implements the upper information integration layer. It connects the data sources to the actual integration and transformation functionality.
- The Data Servers (DS) represent the actual information resources and provide mediated data access for the users.
- The Integration Engine consists of two individual components, the Connector Generator (CG) and the Xcerpt Engine (XE)
    - The Connector Generator generates executable transformations through the composition of modular transformation rules with are retrieved from the rule repository. Global data schemas that define the information architecture determine the generation.

    - The Xcerpt Engine executes the generated transformation based on input data delivered by the Mediation Engine and returns the result to the mediation layer.
- Furthermore, data resources are part of the architecture. These include the transformation rule repository and the data schema repository.

Other elements of the style definition are connectors and properties that constrain the component-connectors structures. The style in Fig. 6 defines connectors between the components, which are outlined in Fig. 5. Essentially, connectors are provided between application services, which perform information integration, between the layers, and for the integration of possibly external data servers. The following properties characterise the style, i.e. the constraints for concrete architectures, in more detail.

- Number restrictions. Except for AS-to-AS connectors and ME-to-DS connectors that allow for multiple application services and multiple data sources, we require 1:1 connector relationships, i.e. do not allow multiple connections to be active between the relevant components.
- Disjointness and completeness conditions. We require both disjointness and completeness with respect to the components, i.e. all components identified must be implemented through separate services (disjointness) and no other components are needed to implement this information and service integration architecture (completeness). All services can be categorised using the provided component types.

Only the adherence to these structural constraints makes an architecture identifiable as an integration architecture and, consequently, enhances maintainability.

Architecture type-level styles for service integration are the counterpart to declarative and modular transformation rules for information integration. The integration style defines structures of a mediator architecture, which are often used in similar application architectures, in a maintainable and reusable form. This style, if implemented, achieves maintainability as follows. Loose coupling between services and layers is the key contributor to maintainability. The decoupling of data integration and transformation service localises the change impact.

The integration style is the core of the architectural solution to maintainable service and information integration. This solution can be improved though the integration of architectural design patterns into the style. Design patterns have been widely used for the design of object-oriented systems and their applicability to services architectures has been investigated. For instance, for the ASP context that we have mentioned as one of our motivating application scenarios, the client-dispatcher-server pattern can be applied to the customer-mediator-data server configuration. The client-dispatcher-server pattern defines components and connections for a broker- or mediator-style system. Other patterns that are more suitable for contexts without the client-provider architecture of ASPs can also be applied.

4.6 Model Refinement and Integration

The context of the template-based integration solution is an incremental, refinement-style development process. An incremental process from the BDM to the IAM level based on the proposed integration techniques is envisaged. We outline the development activities here. The starting point is an integrated model consisting of a business process model and an information

model in the form of an architecture description language. Our approach is, based on the business-level models, to identify architectural entities and to add architectural constraints. We identify the following activities that form steps of the development process:

- Step 1: definition of the architecture constraints based on process and information model elements (e.g. domain models that impose constraints on service structures and attributes).
- Step 2: definition of services with service identification (Step 2a) and service categorisation (Step 2b) based on architecture constraints and reference architectures.
- Step 3: service-based integration service architecture modelling through process composition, based on service integration (Step 3a) and information transformation modelling through rule definition, based on information integration (Step 3b).

The steps 3a and 3b shall briefly be illustrated as they fall within the focus of this investigation.

Step 3a addresses service integration at the integration architecture level. Fig. 7 presents an extension of the initial business process model from Fig. 1, that models the activities as services and add architecture level descriptions such as

- data flow (e.g. withdrawal request),
- separation and distribution of services and processes (e.g. the two partitions for requests and notifications) ,
- inter-component interaction (e.g. from the send reject request and to the execute reject service).

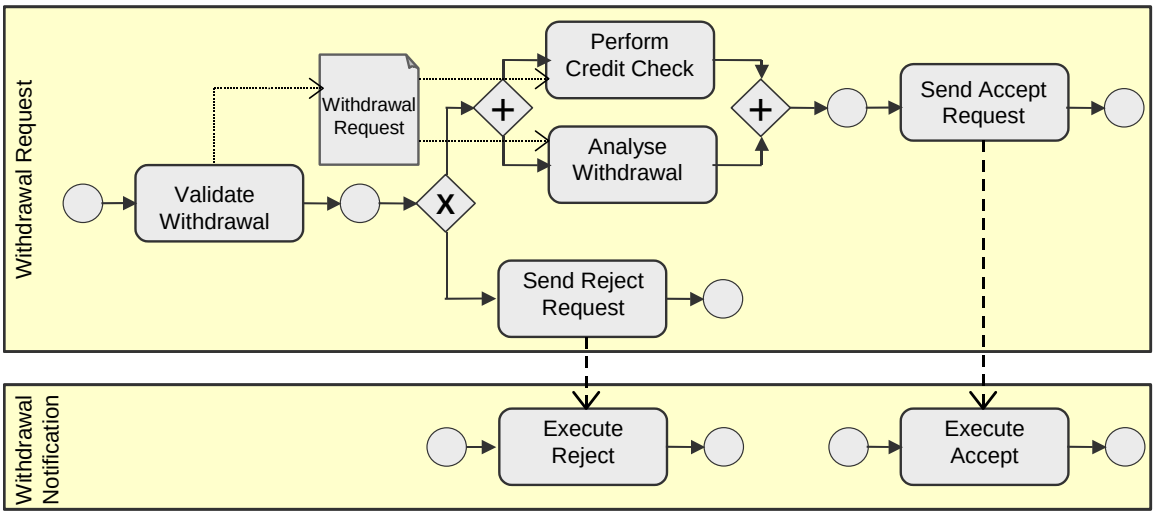

Figure 7. Service integration.

Step 3b is about information integration at the integration architecture level. Fig. 8 extends the initial semantically enhanced information architecture from Fig. 2 by XML representations for heterogeneous data structure for different local application systems or services. Each of these is mapped to the semantic information architecture to ensure consistency and enable automated connector generation.

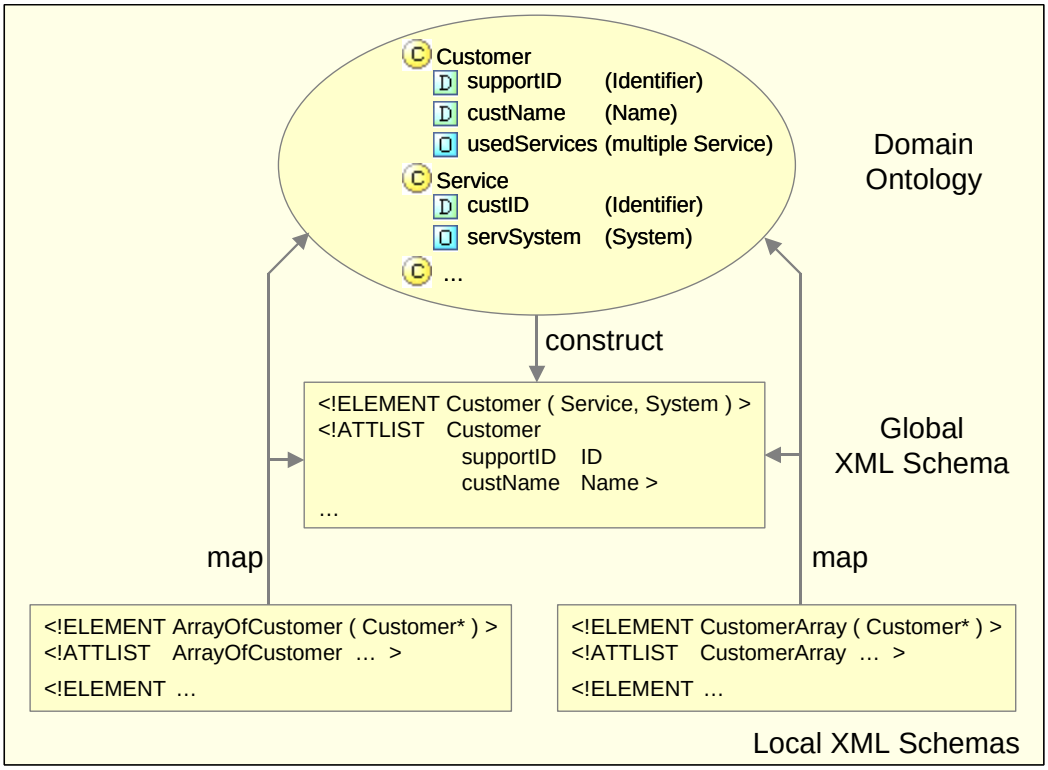

Figure 8. Information integration.

The result is a 2-layered service process integration – an application service layer and a mediation layer consisting of integration and transformation – which is integration style-compliant. This is platform-independent, i.e. it is not yet about orchestration. Declarative and modular transformation rules provide the maintainable information integration.

4.7 Integration Architecture Platform

While our focus has been on the techniques to realise integration at an architectural level, we briefly discuss implementation technologies in order to outline the practicality and feasibility of the techniques. The key platform components needed for an implementation are

- database management systems as schema and rule repositories and
- execution engines for Xcerpt and WS-BPEL to process transformations and application and mediation processes,

all of which as available as open source tools.

The central implementation problem is service orchestration. Service orchestration is about attaching concrete Web services to abstract services (or activity elements that represent services in abstract business processes) [33]. Some applications might provide WSDL-based interface descriptions that can be used to facilitate orchestration. The orchestration language WS-BPEL can be generated from the abstract process modelling notation and the information model through their canonical XML and WS-BPEL representations. The higher-level service processes

are those from the architectural design level that integrate mediator service calls. The transformation to WS-BPEL needs to add mediator calls if necessary.

5. Maintainability Evaluation

The presented information and service-oriented integration architecture is tailored towards improved modifiability and maintainability. We have already discussed individual aspects such as declarative and modular transformation rules, layered architectures and the integration style with respect to their maintainability benefits in the respective sections, but we provide another, comprehensive and scenario-driven evaluation here for our solution. Our solution has been developed based on a number of projects we have been involved in. The following integration scenarios illustrate the scope of our solution:

- migration projects and new developments in the banking and insurance sectors, based on activities of a solution provider that uses an in-house architectural integration framework,
- a large-scale internal integration project based on SOA technologies in the mining sector where human resource and project management had to be integrated across several locations,
- a substantial integration of an ASP beyond enterprise boundaries involving client information access and customer data services, which has provided the setting for the protocol-based evaluation.

These scenarios have also provided us with an evaluation context and evidence about the feasibility of the approach. We discuss the third scenarios here in more detail.

The effectiveness of the proposed integration architecture in terms of maintainability aims shall be evaluated using a widely used evaluation method. The Architecture-Level Modifiability Analysis (ALMA) provides a framework to evaluate software application architectures [3]. Change scenarios are defined and used to elicit and evaluate the modifiability goal. We have evaluated an application service provider (ASP) system after the release of a first prototype implementation of our layered engine-based architecture and have compared it with an existing traditional XSLT-based and ad-hoc WS-BPEL-based ASP solution that has been already in place. An architecture-level impact analysis identifies if an architectural element is affected by a change scenario directly or indirectly. We have defined three scenarios that relate to changes in

1. business rules (clients change the services requested from ASP),
2. data source providers (structural changes in the data provider service architecture),
3. integration rules (caused by data model changes).

The results of the ALMA-based evaluation for the ASP scenarios are presented in Table 2.

Table 2. ALMA-based modifiability analysis of the integration architecture.

Change Scenarios	*Effect on proposed integration architecture*	*Effect on existing architecture*	*How achieved (tactics used in the proposed mediated architecture)*
Scenario 1: business rules	Composite rules	Transformations	Automation: automatic connector construction at runtime
Scenario 2:	Ground rules, maybe	Transformations	Modularity: query part and

data source provider	some immediate rules	and architecture	construct part of an integration rule are separated
Scenario 3: integration rules	New version of composite rules, or reuse or addition of ground and immediate rules	Transformations and architecture	Integration rule repository and independent data services: the connector generator injects no code into integration flow

The impact resulting from each change scenario is only local in our proposed solution, whereas in traditional solutions, the entire transformation set-up including the software architecture can be affected. The declarativity and modularity of the transformation rules and the separation of architectural concerns such as connector generation and execution (which is Xcerpt-specific) from the mediation process as such (which is Xcerpt-independent) into different architectural layers are the contributors to a maintainable solution in our case. The architectural style manifests the architectural benefits of the mediated information and service integration architecture.

There is a trade-off between maintainability and performance. Levels of indirection and dynamic connector generation inevitably decrease performance. Our prototype has, however, demonstrated that these in general do not exceed 15-20% of the overall transformation time, which is acceptable in most ASP situations [36]. Further improvements can be achieved by caching frequently used connectors for standard queries, if necessary.

6. Conclusions

More and more business information systems are implemented and integrated using service-oriented architecture (SOA). Integration is a central problem at several layers of abstraction and for several perspectives. Often, the term IT architecture integration is used to denote the integration of software applications in terms of its information and software component aspects.

We have presented a solution to IT architecture integration that focuses in particular on maintainability requirements. The majority of current solutions use ad-hoc approaches to information and service-level integration. These have the drawback of a lack of structure and comprehensibility, which results in difficult change impact analysis and often prevents changes from being localised. Our techniques are based on

- declarative and modular transformation rules to address information integration
- architectural style and patterns to define a mediate service integration architecture

combined in a coherent integration architecture approach. The aim of this architecture is to support the integration of existing services and service-based architecture as part of an enterprise application integration approach. The benefit of the integration architecture is improved maintainability through improving the determination of necessary changes (based on explicit information and architecture structures) and limiting the impact of changes (based on separation of concerns and loose coupling for both information and service architectures).

Quality of integration architecture is becoming a central concern. In addition to maintainability, other qualities need to be considered. The presented framework already enhances consistency

through its semantic information architecture and the architectural style. Performance is another aspect, which is, however, beyond the scope of this investigation. However, we have demonstrated that performance trade-offs due to improved maintainability are within acceptable ranges. While performance is a system property, automation is like maintainability an aspect of the software process that would need to be addressed as well. Another issue to be looked at is the extension of the scope beyond migration and integration.

References

[1] Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004). Web Services – Concepts, Architectures and Applications. Springer Verlag.

[2] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice* (2nd Edition), SEI Series in Software Engineering, Addison-Wesley.

[3] Bengtsson, P., Lassing, N., Bosch, J. and Vliet, H. (2004). Architecture-Level Modifiability Analysis (ALMA). *Journal of Systems and Software*, 69 (1), pp. 129-147.

[4] Bry, F. and Schaffert, S. (2002). Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: *Proceedings Intl. Conference on Logic Programming*. LNCS 2401, Springer-Verlag.

[5] Conrad, S., Hasselbring, W., Koschel, A. and Tritsch, R. (2006). *Enterprise Application Integration*, Elsevier Spektrum.

[6] Daconta, M.C., Obrst, L.J. and Smith, K.T. (2003). *The Semantic Web – a Guide to the Future of XML, Web Services, and Knowledge Management*. Indianapolis, USA: Wiley & Sons.

[7] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, Y. D., Vassalos, V. and Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*. 8(2), pp. 117-132.

[8] Garlan D., and Schmerl, B. (2006), Architecture-driven modelling and analysis. In: *Proceedings 11th Australian Workshop on Safety Related Programmable Systems* (SCS'06), Conferences in Research and Practice in Information Technology, Vol. 69.

[9] Haller, A., Cimpian, E., Mocan, A., Oren, E. & Bussler, C. (2005). WSMX - a semantic service-oriented architecture. In: *Proceedings Intl. Conference on Web Services ICWS 2005*.

[10] Hasselbring, W. (2000). Information System Integration, *Communications of the ACM*, 43(6), pp. 32-36.

[11] Hess, A., Humm, B. and Voß, M. (2006). Rules for High-quality Service-oriented Architectures (in German), *Informatik Spektrum*, 29(6), pp. 395-411.

[12] Jhingran, A.D., Mattos, D. and Pirahesh, N.H. (2002). Information Integration: A research agenda. *IBM System Journal* 41(4).

[13] Krafzig, D., Banke, K. and Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice Hall.

[14] Lenzerini, M. (2002). Data integration: A theoretical perspective. In: *Proceedings Principles of Database Systems Conference PODS'02*, pp. 233-246. ACM.

[15] Leymann, F., Reisig, W., Thatte, R.N. and van der Aalst, W.M.P. (Eds.) (2006). The Role of Business Processes in Service Oriented Architectures, *Dagstuhl Seminar Proceedings*, Vol. 06291.

[16] Medvidovic, N. and Taylor, R.N. (2000). A Classification and Comparison Framework for Software Architecture Description Languages, *IEEE Transactions on Software Engineering*, 26(1), pp. 70-93.

[17] Object Management Group (2007). *Business Process Modeling Notation (BPMN)*, http://www.bpmn.org/

[18] Orriens, B., Yang, J. and Papazoglou, M. (2003). A Framework for Business Rule Driven Web Service Composition. Jeusfeld, M.A. & Pastor, O. (Eds). In: *Proceedings ER'2003 Workshops*, LNCS 2814, pp. 52-64. Springer-Verlag.

[19] Pahl, C. (2007) Layered Ontology-based Modelling of Service-based Software Systems, *Information and Software Technology*.

[20] Papazoglou, M. P., Dustdar, S., Leymann, F. and Krämer, B.J. (2005). Service-Oriented Computing: A Research Roadmap, Service Oriented Computing (SOC), *Dagstuhl Seminar Proceedings*, Vol. 05462.

[21] Payne, T. and Lassila, O. (2004). Semantic Web Services. *IEEE Intelligent Systems*, 19(4).

[22] Peltier, M., Bezivin, J and Guillaume, G. (2001). MTRANS: A general framework, based on XSLT, for model transformations. In: *Proceedings of the Workshop on Transformations in UML WTUML'01*.

[23] Reynaud, C., Sirot, J.P. and Vodislav, D. (2001). Semantic Integration of XML Heterogeneous Data Sources. In: *Proceedings IDEAS Conference 2001*, pp. 199–208.

[24] Rosenberg, F. and Dustdar, S. (2005). Business Rules Integration in BPEL - A Service-Oriented Approach. In: *Proceedings 7th International IEEE Conference on E-Commerce Technology*.

[25] Rouvellou, I., Degenaro, L., Rasmus, K., Ehnebuske, D. and McKee, B. (2000). Extending business objects with business rules. In: *Proceedings 33rd Intl. Conference on Technology of Object-Oriented Languages*. pp. 238-249.

[26] Schaffert, S. (2004). *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. PhD Thesis, University of Munich.

[27] Seltsikas, P. and Currie, W.L. (2002). Evaluating the application service provider (ASP) business model: the challenge of integration. In: *Proceedings 35th Annual Hawaii International Conference 2002*. pp. 2801 – 2809.

[28] Stal, M. (2002). Web Services: Beyond Component-based Computing. *Communications of the ACM*, 45 (10), pp. 71-76.

[29] Stern, A. and Davis, J. (2004). Extending the Web services model to IT services. *Proceedings IEEE International Conference on Web Services*. pp. 824 - 825.

[30] W3C – the World Wide Web Consortium. (2004). *The Semantic Web Initiative*, retrieved April 21, 2007 from http://www.w3.org/2001/sw.

[31] Widom, J. (1995). Research problems in data warehousing. In: *Proceedings of 4th International Conference on Information and Knowledge Management*.

[32] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, Volume 25. March 1992, pp. 38-49.

[33] World Wide Web Consortium W3C (2006). *Web Services Architecture*, http://www.w3.org/TR/ws-arch/.

[34] WS-BPEL Coalition (2004). *WS-BPEL Business Process Execution Language for Web Services – Specification Version 1.1*, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.

[35] Zhang, Z. and Yang, H. (2004). Incubating Services in Legacy Systems for Architectural Migration. In: *Proceedings 11th Asia-Pacific Software Engineering Conference APSEC'04*. pp. 196-203.

[36] Zhu, Y. (2007). *Declarative Rule-based Integration and Mediation for XML Data in Web Service-based Software Architectures*. M.Sc. Thesis. Dublin City University.

[37] Zhu, F., Turner, M., Kotsiopoulos, I., Bennett, K., Russell, M., Budgen, D., Brereton, P., Keane, J., Layzell, P., Rigby, M. and Xu, J. (2004). Dynamic Data Integration Using Web Services. In: *Proceedings 2nd International Conference on Web Services ICWS'2004*.