Model-driven Connector Development for Service-based Information System Architectures

Claus Pahl
Dublin City University, School of Computing, Dublin 9, Ireland
Email: claus.pahl@dcu.ie

Yaoling Zhu
Dublin City University, School of Computing, Dublin 9, Ireland
Email: yao.zhu3@mail.dcu.ie

***Abstract*—The question whether services can provide a solution for software integration and interoperability problems has been debated. Service-oriented architecture (SOA) now seems to become the most widely used software integration framework. Web services provide the predominant platform for the integration of information systems. A model-driven solution for the development of connectors for information system architectures shall be presented. While most model-driven approaches focus on software components, we investigate system integration through a model-driven connector development approach for the context of service-oriented architectures. Maintainability and automation requirements are discussed in relation to integration and architecture aspects.**

***Index Terms*—service-oriented architecture, information integration, model-driven development, software architecture, semantic data modelling, connectors, mediated architecture, declarative data transformation.**

I. INTRODUCTION

Whether services can provide the solution for software integration problems is a question that has been debated for a while. Service-oriented architecture (SOA) seems to become the most widely used software integration framework [1]. Web services provide the predominant platform for SOA [2]. SOA as an integration solution supports a number of application scenarios ranging from XML data integration to specific context such as application service providers to recent advances in platform applications such as service-based data grids. SOA and services can provide technologies to improve information integration.

While the feasibility of a service-based solution for information systems integration has been widely addressed [3,4,5], the focus needs to shift towards quality in order to provide cost-effective and reliable solutions for organisation. Services provide the necessary interoperability needed for integration problems, but quality aspects such as maintainability, cost-effectiveness, or consistency require advanced solutions to be used as part of a SOA integration methodology. Our aim here is to discuss integration techniques based on connectors as mediators between providers and consumers of information. We aim to demonstrate that progress beyond standard solutions and technologies with their limitations in terms of automation and maintainability – and resulting cost and reliability problems – is possible [6].

Model-driven development (MDD), promoted by industry bodies such as the OMG, proposes an approach that, although not specific to the information systems and services context, can provide a framework for our investigation [7,8]. MDD focuses on maintainability through model-centricity and on automation of programming activities through code generation. MDA emphases automation and encourages model reuse.

Architectural approaches are often based on a component-and-connector view of systems. Connectors are the key elements of integration architectures. The idea of model-driven development for integration is to use a semantics-driven approach to define integration rules that act as models of connectors for service integration. Declarative rules shall be used as models of connectors. The latter are derived from the models and will actually implement information integration. Abstraction is the core principle of modelling and generating connectors through abstract integration rules. The main distinction of our approach compared to other model-driven development is our focus on connectors, not components. Thus, integration models in the form of transformation rules (rather than the usual UML-based class and component diagrams) are the basis of this investigation.

This investigation starts by motivating the problem from the specific perspective of an application service provider scenario in Section II. Then, the information integration problem and its link to SOA in the context of related work is defined in Section III. Section IV presents a model-driven integration solution in terms of three components: data integration, mediated architecture, and a process model. Semantically enhanced mediation is considered as an approach to further enhance the solution in Section V. This solution is evaluated in the context of

Manuscript received August 11, 2008; revised December 2, 2008; accepted January 10, 2009.
Corresponding author: Claus Pahl (claus.pahl@dcu.ie).

maintainability requirements in Section VI. We end with a discussion of related work and some conclusions.

II. Information Integration in SOA Environments – Principles and Requirements

Information integration is the problem of combining heterogeneous data residing at different sources in order to provide the user with a unified view [9,10]. Information integration is central to meaningfully and consistently adapt services and the underlying data sources to specific client and provider needs. A common understanding through agreed and formalised semantics and mappings between different syntactical and structural representations is essential. The aim of information integration is to define mappings between the individual data sources and a unified view of these data sources.

These definitions of mappings and views are formulated in terms of query and transformation languages [5] Transformation languages provide the solution to the first component of our solution – information integration [4,11]. In order to find a solution that meets the quality criteria, here in particular maintainability as a consequence of expected high degrees of change and evolution, a model-driven approach shall be considered.

A. Information Integration for ASPs

The Application Service Provider (ASP) business model promotes the use of software as a service [12]. An example of this model is information systems outsourcing, i.e. the handing over management of an enterprise's information infrastructure to a third party. The ASP takes responsibility for managing the software application on its own infrastructure. The ASP maintains the application and ensures that system functionality and data are available when requested. The ASP context is the environment in which a solution is developed and evaluated its characteristics.

Recently, service-based platforms are used to provide integration solutions for ASP applications. SOA-benefits for ASPs are interoperability and dynamic configurability. Specifically, change and evolution problems are omnipresent in these information systems due to a large number ASP clients, services offered by an ASP and data associated to services. Information architectures create specific needs in terms of schema evolution, business processes changes, and participant (provider, client etc.) changes. Changes in data representations and consequently in integration rules can be expected frequently. Maintainability is therefore a particular focus of this investigation.

At the core of service-based integration architecture is an information integration problem. Provider and consumer in an ASP context might internally use different data schemas, as the example of a customer representation in Fig. 1 demonstrates. The ASP defines the central model, onto which customer models are mapped. Transformations can still occur in both directions. A transformation approach is therefore a core element of an integration solution. Information integration needs to be mediated in service architectures using connectors, which is the second problem in this context. Data schema integration cannot be fully automated – the syntactic representation of data schemas does not completely convey semantics. Consistency can only be achieved manually, which leads to semantically enhanced information architectures as the third problem that we investigate.

In summary, the information integration requirements in service-based architectures entails a number of specific needs: automation and dynamic integration to increase flexibility and maintainability of integration and mediation in large-scale applications. The model-driven development of integration in the form of transformations can address these requirements.

B. Transformation Principles and Requirements

Transformation is at the core of an integration solution. Two types of transformation languages exist – procedural ones, which are mainly used today, and declarative ones, which promise better quality through abstraction. XSLT is the predominant representative of procedural languages, but suffers from a number of drawbacks:

- procedural XSLT transformations are difficult write and read due to an interleaved specification of query and construction elements,
- XSLT transformations are difficult to reuse as a consequence of a lack of clear structure and limited intuitivity,
- the syntactical integration of query and construction part often prevents easy changes and consequently hampers maintainability for large-scale information integration.

The benefits that apply to declarative query languages (for instance, consider the success of SQL as a declarative query language) also apply to transformation languages, which are part query, and part information construction language. A variety of declarative transformation languages – such as ATL, QVT, or Xcerpt – can alleviate this problem. Declarativity enables a higher degree of abstraction – a central feature of MDD. A selection of procedural and declarative languages is compared later on, after determining the SOA-specific requirements for such a language in this context.

The requirements for an information integration solution for SOA-based information systems can based on [13] be summarised as follows:

- declarative transformation achieves a higher level of abstraction resulting in improved reusability and maintainability through abstraction from operational aspects and easier understandability,
- modular transformation specification (beyond the separation of query and construction part) is needed to achieve a higher degree of maintainability,
- orchestration of the transformation process – embedding of transformations into a mediator workflow specification – is necessary in order to enhance maintainability through separation of concerns and loose coupling.

Executable models of these transformations will actually be the central solution of our integration solution.

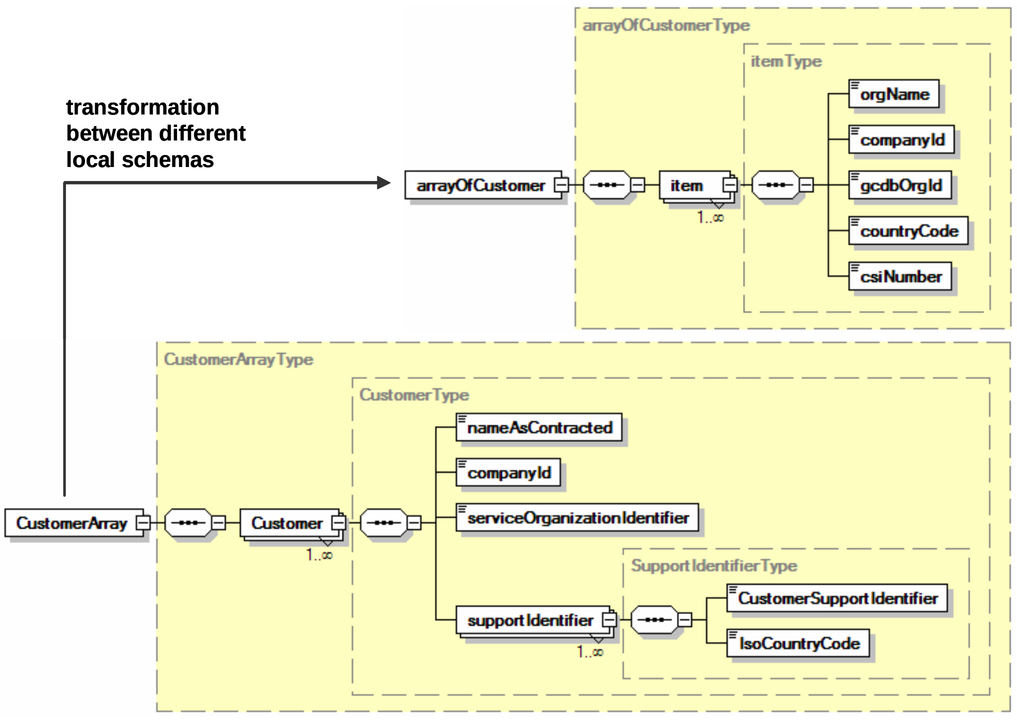

Fig.1. Data Integration and Transformation Problem.

C. *Query and Transformation Languages*

A number of transformation languages [9,11,14] including XSLT, XML-QL, Xquery and Xcerpt shall be surveyed to determine their suitability to provide a maintainable model-driven integration solution. The following criteria are applied [13] and form the basis of the comparison in Table 1:

- general aspects: data model, language, application domain (i.e. whether data structures can be defined, which data representation languages are supported, and in which context is the transformation to be applied)
- structure: query/construct separation, answer as query (i.e. whether query and construction elements are separated and queries can be returned as results)
- expressiveness of query language: join, incomplete queries, new elements, tag variables, nesting, grouping, pattern matching (i.e. support of a range of query and transformation language-specific features)
- analyses: cyclic terms, query reduction (i.e. whether specific analyses and optimisations take place);
- usability: ease of use, modifiability effort, tool support (i.e. criteria relating to practical considerations regarding the use of the tool).

These criteria address the necessary abstraction mechanisms and analysis techniques to improve maintainability, but which also affect automation and performance. Pragmatic concerns such as tool support are considered as well. The result of this survey of six languages – three procedural and three declarative – is shown in Table 1.

The shortcomings of the widely used procedural languages in the context of the given requirements and the language comparisons have led us to choose a declarative language. All declarative languages score well. Xcerpt [15] satisfies the major criteria. Xcerpt is chosen as it provides stable, open-source based tools support for efficient and modular transformations. It is specifically suited for Web-based information systems through its support of not only XML, but also specifically Semantic Web languages such as RDF and OWL.

While Xcerpt is an ideal candidate, other recently developed and well-supported transformation languages such as ATL and QVT are similarly suitable candidates. While e.g. QVT satisfies the criteria, it is currently not as well supported through tools and accessible tutorial material as Xcerpt. Xcerpt extends the pattern-based approach, which is used in other query and transformation languages, in following aspects:

- Firstly, query patterns can be formulated as incomplete specifications in three dimensions. Incomplete query specifications can be represented in depth (which allows XML data to be selected at any arbitrary depth), in breadth (which allows querying neighbouring nodes by using wildcards) and in order. Incomplete query specifications allow patterns to be specified in a more flexible manner, but without losing accuracy.
- Secondly, the simulation unification computes answer substitutions for the variables in the query pattern against underlying XML terms – similar to language UnQL, but only strict unification is used in UnQL.

Xcerpt provides a runtime environment with an execution engine at its core [16]. The central problem that we discuss here is how to embed this type of environment, which can also be found for other query and transformation languages, into a dynamic, mediated service setting.

III. RULE-BASED CONNECTOR GENERATION

Equipped with the central requirements and Xcerpt as the choice as the transformation language, our model-

TABLE 1. Transformation Languages for Model-driven Information Integration for SOA.

Criteria	*XML-QL*	*XSLT*	*XQuery*	*Xcerpt*	*ATL*	*QVT*
Specific data model	Yes	Yes	Yes	Yes	Yes	Yes
Result language	XML	XML	XML	XML	XML	XML
Application domain	Web Data Integration	Generic Transform.	(Web) Data Integration	Web or Semantic Web	Model Transform.	Model Transform.
Query/construct separation	No	No	No	Yes	Yes	Yes
Answer as query	No	No	No	Yes	Yes	Yes
Joins	No	No	Yes	Yes	Yes	Yes
Incomplete query specification	Yes	Yes	Yes	Yes	Yes	Yes
Construction of new elements	Yes	Yes	Yes	Yes	Yes	Yes
Tag variables	Yes	No	No	Yes	Yes	Yes
Nested queries	Yes	Yes	Yes	Yes	Yes	Yes
Grouping	Yes	No	Yes	Yes	Yes	Yes
Pattern-based queries	Partly	No	No	Yes	Yes	Yes
Halt on cyclic query terms	N/A	N/A	N/A	Yes	Yes	Yes
Query reduction	No	No	No	Yes	No	Yes
Ease of use	Yes	No	No	Yes	Yes	Yes
Modifiability effort	Low	High	High	Low	Low	Low
Tool support	Little	Sufficient	Sufficient	Sufficient	Sufficient	Little

driven technique for service-based information systems integration shall now be presented. The two solution components – integration models and mediated architecture – are presented in the following subsections. The conceptual overview in Fig. 2 illustrates the solution principles. A semantically enhanced information architecture (which will be addressed in Section IV) defines the information model. Declarative transformation rules, consistent with the information model, are abstract models of integration between local data schemas. From these, executable connectors can be generated. These connectors perform mediation in the service-based architecture. This framework bears similarity to the OMG-supported Model-Driven Architecture (MDA). The information architecture is a computation-independent model, whereas the integration model is platform- and execution-independent. Only the connectors are platform-specific executable models.

Note, that we can distinguish two types of transformations in our discussion:

- data integration, i.e. data transformation using Xcerpt, is integration-oriented transformation,
- rule to connector generation, i.e. the model-driven development activities, is generation-oriented transformation in the MDD sense.

Transformations for data integration are here the subject of model-driven transformations.

A. Declarative Transformation Rules as Integration Models

Xcerpt [15] is designed for querying and transforming standard Web data (XML, HTML) and Semantic Web data (RDF, OWL). The following design principles can be distinguished: declarative transformation rules, the separation of matching and construction part, and goal-based query programs and transformation rules [16]. Fig. 3 illustrates a goal-based query program for the customer example from Fig. 1 that shows the separation of query and construction part. In addition to these query programs, which include references to input and output data, Xcerpt distinguishes transformation rules (without references) from goal-based queries.

Each rule is a declarative, abstract model of an integration-oriented transformation. In order to enhance the solution, we aim to define focussed, modular rules. A layered approach to transformation rule specification achieves compositionality of rules and consequently the required modularity of rule definition:

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

Based on a user query, a composed connector based on several individual rules can then be generated from individual rules (stored in a rule repository) and executed automatically.

B. Model-based Connector Generation

Connectors are the integration facilitators. They mediate between customer queries/updates and the information provider. They can be generated automatically from the rule models, determined by schema definitions and the user query. A customer query could be the following:

```
select Customer
```

```
  from   CustomerArray
  where  Service = "..."
```

A connector is based on a goal (essentially the query), which would involve two rules that Rule 1 in Fig. 4 refers to – Customer and Service. This results in an instruction to get the corresponding data elements from the resources for customers and services. We illustrate this using an operational pseudo-code formulation. Two stages – connector preparation and connector execution – can be distinguished.

A first stage is a connector preparation based on references Ref_C and Ref_S to resources for customers and services:

```
Rule_C :=
  retrieveRule ( translRule ( Customer ) ),
Rule_S :=
  retrieveRule ( translRule ( Service ) ),
Con :=
  connectorGenerate ( C, S, Ref_C, Ref_S,
                              Rule_C, Rule_S)
```

At the second stage, a connector based on previously generated connector Con and retrieved data Data_C and Data_S corresponding to the reference is executed:

```
Res := connectorExe ( Con, Data_C, Data_S )
```

The repository contains the connector models in the form of abstract rules – without the concrete resources identifiers as in Fig. 3. The connector generator replaces these by concrete references Ref_C and Ref_S. Connectors consist of configured (instantiated) rules needed for the transformation. The second stage of the generation process combines the rules through chaining during the connector execution. It accesses the previously retrieved data from the resources and applies the transformation to it. The generated orchestrated process that invokes the connector generation and execution is discussed further below. Res is the result, which is returned to the user. Note, that the final executable connectors are only generated internally by the Xcerpt engine and executed on the fly. The connector generation here (a wrapper) is a pre-generation and pre-execution configuration of the connector. The two stages reflect two separate invocations by the coordinating integration process. The second connector generation stage shall be detailed now.

Backward goal-based rule chaining is applied to compose rules, which means that variable bindings of constituent rules are chained to the query program itself, i.e. that data is constructed bottom-up through recursive rule application. Fig. 4 presents an example of four layered rules where rule 1 is refined by rules 2a and 2b and rule 2b in turn is based on rule 3. Using this modular approach to rule definition, any potential change to rules to a specific element of the original data models is local.

The transformation rules are stored in a repository. Rules are retrieved from the repository, combined and backward chaining is applied to retrieve, combine and assemble data for specific transformation requests. A connector – an executable transformation that translates between data source and a customer query – is then generated based on a top-level transformation goal. Connectors can be generated on the fly based on layered rules stored in the repository and input data services that provide the required data. The bottom-level rules contain logical references to data sources, which are translated into data service calls during the connector generation. For mediator architectures, rules need to be decoupled from the data resources. In this way, a separation of the application logic, represented in the rules, and the implementation-specific locations of data services is achieved. This separation improves maintainability.

Connector generation means to generate an orchestrated transformation flow by composing a query with the corresponding transformation rules and associated data service calls. The Xcerpt runtime engine reads XML data from data servers and populates them into data terms before processing the transformations[16]. Our wrapper mechanism in the preparation replaces logical identifiers by service calls. The runtime engine is a Web services and is called by the wrapper, which is part of an integration process that controls the mediation.

Two executables are generated from the models:

- A composite goal-based transformation rule for the transformation engine is prepared by the connector generator, as already discussed. The wrapper prepares a customised transformation for the Xcerpt engine, which takes chained rules as input. The actual combined rule is created within the transformation engine. The connector has been described in Section III.B.
- Additionally, an integration process to coordinate the mediation is generated by a query component for the integration process. The activation of the transformation engine is part of this integration process. The integration flow shall later be implemented as a Web services process. A pseudo-code representation for the integration flow is:

```
receive ( ResourceIDs, Query)
  concurrently
    Con  := activate connectorGenerate
    Data := retrieve (ResourceID)
    Res  := activate (
      connectorExe ( Con, Data )
    return Res
```

While the application rules need to be kept in a changeable format, we have implemented these system-level transformations using Java XML-processing APIs for performance reasons.

C. Connector-based Mediated Integration Architecture

The model-driven generation of automated data transformation needs to be implemented adequately in the context of the service platform [17,18]. Different architectural approaches for a target platform for model-driven information integration exist:

- Data warehousing: an in-advance approach that gathers data from the appropriate data sources to populate entities in the global view of the data

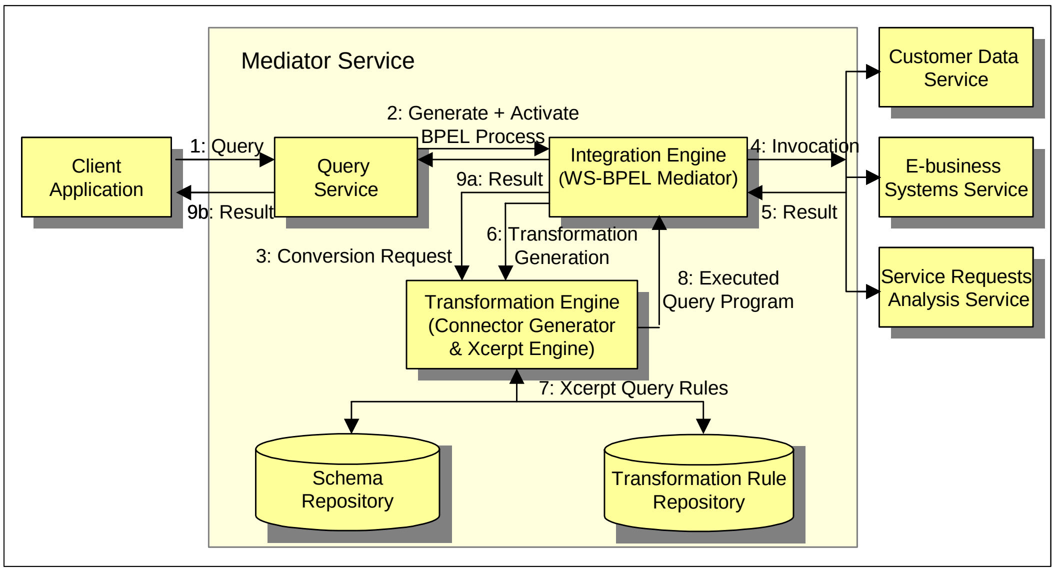
Mediator Service
Client Application
1: Query
9b: Result
Query Service
2: Generate + Activate BPEL Process
9a: Result
Integration Engine (WS-BPEL Mediator)
4: Invocation
5: Result
Customer Data Service
E-business Systems Service
Service Requests Analysis Service
3: Conversion Request
6: Transformation Generation
8: Executed Query Program
Transformation Engine (Connector Generator & Xcerpt Engine)
7: Xcerpt Query Rules
Schema Repository
Transformation Rule Repository

Fig. 5. Mediator Architecture for Service-based Information System Integration.

warehouse. Data, its aggregations and analytic data is stored. Wrappers translate between user and warehouse data formats.

- Federated schema systems: also an in-advance approach based on the early agreement on common schemas and data population. These systems aim to deal with data integration in distributed, autonomous systems. A federated schema defines the integrated view of individual export data schemas provided by the participants.
- Mediated approach: extracts only meta-data from export schemas in advance. View-specific wrappers provide access dynamically. A data merge engine handle updates and ensure consistency according to a global schema.

Flexibility requirements relating to change and maintenance make the mediated approach ideal for service-based information systems [5,19,20] 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. The mediation process itself can here be dynamically generated and customised from the integration models.

The mediated integration architecture shall consists of the following components (Fig. 5):

- data sources provided as XML data services,
- a integration engine that implements the generated orchestrated mediation process,
- a transformation engine consisting of a connector generator that composes executable transformations from queries and stored transformation rules and an Xcerpt transformation engine that carries out the transformation by executing the connector,
- repositories for schemas and transformation rules.

Model-based generated connectors are variable, dynamic elements of the architecture located in the transformation engine, but as we already said, the integration flow for the integration engine is also dynamically generated. The query component configures the integration process, e.g. determines the data servers to be included in the process. The architecture components and their connectivity and interactions are presented in Fig. 4. Three sample services of a typical ASP scenario, part of a customer management facility, are include for illustration.

The runtime behaviour of the architecture is summarised in nine steps (annotated in Fig. 4) that describe the interaction between client, mediator and data servers, but also internal interactions within the layered mediator. The interactions subsume data aggregation (lower mediator layers) and higher-level information processing.

1. Client Application invokes the Query Service.
2. The Query Service generates and activates a WS-BPEL Mediation Process in the Integration Engine.
3. The Mediation Process calls the Connector Generator (Transformation Engine) to construct the connector based on rule and schema models in the repositories.
4. Data Service providers are called by the Mediation Process in the Integration Engine.
5. The data is assembled by the Mediation Process.
6. The data is passed to the Transformation Engine.
7. The data is composed with the connector and executed by the Xcerpt engine.
8. Transformation results are transferred back to the Mediation Process.
9. Results are transferred back to the Client via the Query Interface.

The data aggregation process, i.e. the execution of complex queries in the integration service with connector generator and Xcerpt engine in the lower part of the mediator diagram is separated from the higher-level information processing through the BPEL transformation engine. This results, together with the modular and declarative transformation specification, in improved

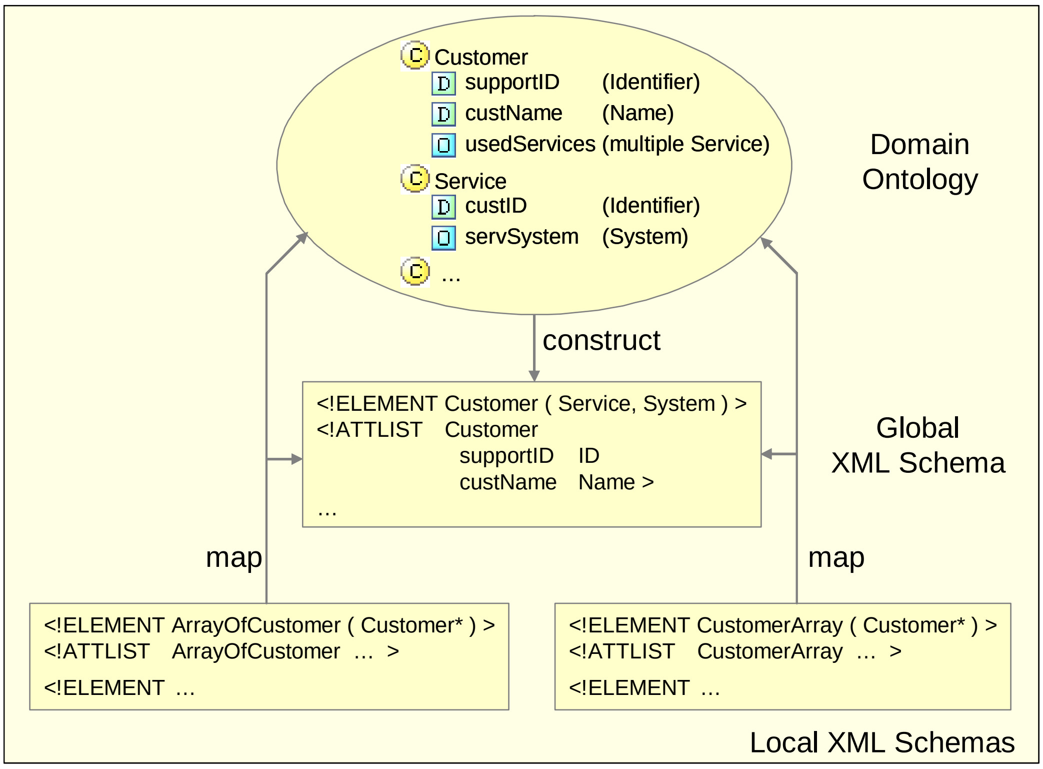
Customer
supportID (Identifier)
custName (Name)
usedServices (multiple Service)
Service
custID (Identifier)
servSystem (System)
...
Domain Ontology
construct
<!ELEMENT Customer (Service, System) >
<!ATTLIST Customer supportID ID custName Name >
...
Global XML Schema
map
map
<!ELEMENT ArrayOfCustomer (Customer*) >
<!ATTLIST ArrayOfCustomer ... >
<!ELEMENT ...
<!ELEMENT CustomerArray (Customer*) >
<!ATTLIST CustomerArray ... >
<!ELEMENT ...
Local XML Schemas

Fig. 6. A Semantic Information Model for the Customer Example.

maintainability and also scalability of the architecture. We discuss the benefits, but also potentially negative implications, in the evaluation (Section V).

IV. Transformation Rule Generation

Automation of model-based generation and consistency between abstract and concrete models are two central aspects of model-driven development. A semantic model of the information architecture is the solution to enhance the two aspects further [21,22]. The use of domain ontologies shall be suggested to define a knowledge-based information model to constrain the integration model and make the transformation rules consistent with the data models. These aims at enhancing the representation of schema-based information architecture to an ontology-based model [3,23]. The benefits of semantically defined information models include:

- a higher degree of semantic integration and consistency through abstract semantic models,
- a higher degree of automation and reliability through formal definitions and automated generation.

A. Ontology-based Semantic Information Model

Ontologies are knowledge representation frameworks about a domain in terms of concepts and properties of these concepts [24]. Ontologies are often defined as conceptualisations of a domain. The Web Ontology Language OWL [25], the main Semantic Web ontology language, is the ideal candidate for semantic annotation in Web-enabled information systems. In order to avoid the verbosity of the XML-based OWL in this investigation, however, an ontology-based representation akin to the proposed Manchester syntax for OWL is used. The `Customer` data type can be semantically defined:

```
Customer =
      exists supportID . Identification and
      exists custName . Name            and
      exists usedServices . Service
Service =
      exists custID . ID                and
      exists servSystem . System
System =
      exists hasPart . Machine
```

Each concept, like `Customer`, is defined in terms of its properties, such as `supportID`. These properties associate information of a specific type or another concept to a given concept – `supportID` assigns an `Identification`, `usedServices` another concept called `Service`. This description defines a semantic information model, represented graphically in Fig. 6.

B. Model-based Transformation Rule Generation

This ontology model opens the opportunity to automatically generate further integration components, i.e. data schemas and transformation rules. Both are central ingredients for the connector generation, Fig.5. A canonical XML schema can be automatically derived from this model, as the following example for `Customer` demonstrates:

```
<!ELEMENT Customer ( Service, System ) >
```

```
<!ATTLIST Customer
             supported  Identification
             custName   Name >
```

The global integration schema is now defined as an abstract and interchangeable domain model. Any local schemas are defined by mappings into either a global XML schema or directly into the ontology model.

In addition to the canonical XML schemas, more importantly also transformations between local schemas can be derived automatically from the ontology model and the schema mappings. A prerequisite is that each of the local schemas is mapped to the domain ontology. A notion of consistency needs to be defined to express semantics preservation.

While a basic solution for this transformation can easily be generated based on the semantic integration of all data aspects, an adequate transformation generation that meets the quality requirements needs to consider the previous transformation requirements (see Section 2.2) such as modularity. A rule construction algorithm from the information model to the integration mode layer is proposed that creates modular rules, similar to those previously discussed. For each concept in order to localise change impact the following is done:

1. define one construction rule per concept of the target schema (based on the concepts from the overarching ontology),
2. identify semantically equivalent concepts in the source schema based on the ontology (local elements map to the same ontology concept),
3. for each concept, determine attributes and copy their counterparts from the source schema (preserves concept properties).

This defines consistent, i.e. semantics-preserving transformations due to the semantic and not only syntactic integration of data. Applied to the structural formulation of the customer schemas in Fig.1, this means that for example the `CustomerArray` and `Customer` rule in Fig. 4 can be automatically generated.

V. Evaluation

The model-driven connector solution has emerged from a number of projects we have been involved in:

- migration projects and new developments in banking and insurance domains, based on activities of a solution provider that uses an in-house architectural integration framework,
- large-scale internal integration projects based on SOA technologies in the mining sector, here human resource and project management applications have been integrated across heterogeneous locations,
- an integration project across enterprise boundaries for an application service provider, involving client information access and customer data services.

These projects, which illustrate the scope of the solution, have also provided us with an evaluation context and evidence about the feasibility of the approach. We evaluate our approach in the context of the third project for which we developed a software prototype.

A. Prototype Implementation and Case Study Evaluation

The overall evaluation is based on a prototype of the architecture that has been implemented as part of a case study. Besides the Xcerpt runtime engine – an open source application – Oracle's BPEL Process Manager and an Oracle database server have been used to implement the service architecture. Although ontology technology in general is less mature than transformation and integration technology, ontology representation and rule construction can be implemented based on two platform techniques and tools. Ontology representation can be addressed with OWL as the language. Protégé tool is a suitable ontology editor (protege.stanford.edu) that can build, populate, translate and export ontologies. The rule construction can be based on the Jena ontology processing API (jena.sourceforge.net), which works on RDF graphs and allows us to process attributes and properties by traversing the ontology.

The case study is part of the ASP's Customer Intelligence Framework (CIF). CIF provides portal-based access for customers and managers to a reporting system for the ASPs on-demand services. The reporting system consists of analyser objects that access content in the form of requests logs, outage tracking data and customer life cycle information. The aim is to support ad-hoc, cross-content adaptable user queries.

The ASP case study application, which was implemented using the model-driven connector development technique based on the prototype, has focussed on the customer lifecycle management system (CIF) with services such as customer data services and service request tracking and analysis (see Fig. 5, right-hand side), made available by an application service provider. This case study has been supported the feasibility investigation and has been used to evaluate the maintainability benefits of the MDD approach – see [13].

B. Maintainability Evaluation – Connector Generation

The presented model-driven integration technique is tailored towards improved modifiability and maintainability. The effectiveness of the proposed technique in terms of these aims shall now be evaluated. The Architecture-Level Modifiability Analysis (ALMA) provides a framework to evaluate the proposed architecture and its development [26]. Change scenarios are used to elicit and evaluate the modifiability goal. Our technique has been evaluated after the release of a first prototype and has been compared with a traditional XSLT-based integration solution. An architecture-level impact analysis identifies if an architectural elements is affected by a change scenario directly or indirectly. The three scenarios relate to changes in business rules (clients change the services requested from ASP), data source providers (structural changes in the data provider service architecture), and integration rules (caused by data model changes), respectively:

- Scenario 1 (business rules): changes affect composite rules in the proposed architecture, whereas the entire XSLT transformation would be affected in the traditional architecture. The improvement is achieved through automation. Automatic connector construction at runtime reduces human intervention.

- Scenario 2 (data source provider): changes affect ground rules and maybe some intermediate rules. Again, the entire XSLT transformation file would have to be changed in the traditional set-up. The benefits are achieved through modularity. Query part and construction part of an integration rule are separated.
- Scenario 3 (integration rules): changes affect the new version of composite rules, or reuse or addition of ground/immediate rules. In the traditional architecture, the entire XSLT transformation needs changing. The integration rule repository and independent data services are the success factors. The connector generator injects no code into the integration flow.

The impact resulting from each change scenario is local in the proposed solution, whereas in traditional solutions, the entire transformation can be affected. The declarativity and modularity of the transformation rules and the separation of connector generation and execution (which is Xcerpt-specific) from the mediation process as such (which is Xcerpt-independent) are the contributors to a maintainable solution in this case.

C. Consistency and Automation Evaluation – Rule Generation

The semantic enhancement clearly improves the degree of automation – a key aim of model-driven development – through in particular automated rule generation and the resulting benefits of only having to modify modular models if changes are required. Semantic enhancements also improve consistency through explicit semantic specification and the fact that consistent, i.e. semantics-preserving transformations are generated automatically.

The trade-off for automation is mostly visible in terms of a preparation overhead for the models. The following models need to be defined in advance: the domain ontology (by all participants) and the mappings from local schemas to the ontology (by the owner of the schema). Although this requires the involvement domain experts and knowledge engineers and creates some initial start-up costs, this is usually a once-off activity and can be expected to be amortised soon. The benefits and drawbacks can therefore be summarised as follows. An in-advance preparation enhances the effectiveness of solutions in general. The transformation generation enhances flexibility, but also decreases efficiency if performed dynamically.

D. Discussion

The previous subsections have demonstrated the benefits – both connector and rule generation – in terms of maintainability, consistency and automation. However, the effects on performance and development costs might be detrimental. There is often a trade-off between maintainability and performance in software. Two factors decrease performance in our solution:

- levels of indirection caused by the architectural separation of mediation, connector generation and transformation,
- dynamic connector generation based on rules and schemas stored in the respective repositories.

Our prototype has, however, demonstrated that these two factors together in general do not exceed 15-20% of the overall transformation time compared to the traditional architecture, which is acceptable in most ASP situations. The platform we are using – our system is based on Oracle‘s BPEL Process Manager – is comparatively efficient and does not cause additional negative effects.

Another issue are the development costs. We can estimate the costs for a long-term solution based on the costs of the prototype and the development of the rule repository. This can be addressed through an incremental process, which makes the development systematic and predictable.

VI. Related Work

A number of related approaches exist, in particular in the context of model-driven middleware [27,28,29], but no specific model-driven integration solution based on a layered model-driven generation approach (as outlined in Fig. 2) has been presented. A number of technical solutions for rule-based integration and mediation exist, though. Many data mediation systems adopt logic rules to express the correspondences between the schemas. In general, logic rules elicit schema information such as element names, schema structures and integrity constraints. The rule-based approach provides the following advantages compared to other approaches. Firstly, declarative rules tend to be generic. Secondly, these rules are intuitive to learn for users and inexpensive to use for data integration systems. Thirdly, declarative rules are also useful to derive new rules to create new elements in the integrated schema in an automatic manner.

MSL (Mediator Specification Language) introduced in the MedMaker framework is a declarative rule-based language to generate the mediators based on the declarative specifications. The rules or specifications can be queried by MSL (Mediator Specification Language). MSL is powerful enough to carry on operations such as grouping from one source object, removing redundancies and removing inconsistencies. These rules are declarative, rather easy to understand and provide a high level of abstraction. These rules are mainly deduction rules. If the query pattern in the rule body is satisfied, then the construct pattern in the rule head is assumed to hold. Usually, the construction pattern uses data selected in the query pattern. While similar in terms of its capabilities, MSL is not focussed on dynamic generation and is not supported by a semantic information architecture. Ref. [3] provides the latter, but is not specifically maintainability-optimised.

A declarative, rule-based approach has also been applied to the data transformation problem by [4] and [11]. The problem is its integration into a service-based composition. In [4], the data integration engine is built in WS-BPEL, the composition schema are in activity diagram as its business logic and the components invocation orders are predefined in the composition schema, orchestrations are defined by specifying which operations to invoke from the beginning of the execution to the end. The business logic in our approach is defined as business rules that govern the data integration process. The data integration rules are generally elicited from the business logic. The common information model governs what types of services and components are involved in the composition. Ref. [30] presents an approach for automated connector generation. However, their approach is based on state machines, which we feel does not provide the right level of abstraction in order to be suitable in a collaborative development context for enterprise-oriented information systems, as we envisage.

Compared to these, and others such as [3], our technique demonstrates the maintainability of a layered, i.e. modular and loosely coupled integration architecture. The composability of modular rules enhances their reusability. Our model-driven architecture technique decouples mediation from integrations, which makes not only data services, but also the transformation engine replaceable.

VII. Conclusions

Many information systems are now modified in their often complex infrastructures to manage and integrate information using services, which illustrates the importance of quality in integration solutions. Service-oriented architecture is a promising approach to the integration problem, but the complexity of information architectures combined with the constant change and evolution in this context requires SOA and service platform technologies to be tailored to deliver cost-effective and reliable solutions. A model-driven development approach can deliver this quality-enhanced solution.

MDD succeeds here in providing a maintainable integration solution as only the information and transformation models are changed and the executable integration components can be generated automatically. This confirms the reputation of MDD benefits, including improved maintainability. The presented model-driven integration technique utilises semantic information integration models in SOAs. A declarative style of transformation, in an extension based on an ontology-based information model, with automated, dynamic transformation creation is a central solution component. The model-driven development of a flexible mediator process and connectors is crucial for consistency, automation and maintenance.

Integration efforts are part of a wider re-engineering and migration strategy. Problems of re-engineering and the integration of legacy systems are aspects that have not been addressed here specifically. The ASP example is a typical example of legacy systems integration into service-based architectures. The introduction of data transformation techniques for re-engineering activities can improve the process of re-engineering legacy systems and adopting SOA to manage the information technology services. This, however, is an aspect that remains to be investigated in detail. Nonetheless, in the context of mostly component-focussed MDD approaches, model-driven integration and connector generation complements these efforts.

References

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

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services – Concepts, Architectures and Applications*. Berlin: Springer Verlag, 2004.

[3] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "WSMX - a semantic service-oriented architecture", in *Proceedings Intl. Conference on Web Services ICWS 2005*. 2005.

[4] B. Orriens, J. Yang, and M. Papazoglou, "A Framework for Business Rule Driven Web Service Composition", in *Proceedings ER'2003 Workshops*, LNCS 2814, pp. 52-64. Springer-Verlag, 2003.

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

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

[7] Object Management Group, *Model-Driven Architecture MDA Guide V1.0.1*, OMG, 2003.

[8] B. Selic, "The Pragmatics of Model-Driven Development". *IEEE Software*, vol. 20(5), pp. 19-25, 2003.

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

[10] G. Wiederhold, "Mediators in the architecture of future information systems". *IEEE Computer*, vol. 25, pp. 38-49, March 1992.

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

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

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

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

[15] F. Bry and S. Schaffert, "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, 2002.

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

[17] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, Y.D. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS approach to mediation: Data

models and languages", *Journal of Intelligent Information Systems*, vol. 8(2), pp. 117-132, 1997.

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

[19] A. Stern and J. Davis, "Extending the Web services model to IT services", in *Proceedings IEEE International Conference on Web Services*, pp. 824 – 825, 2004.

[20] J. Widom, "Research problems in data warehousing", in *Proceedings of 4th International Conference on Information and Knowledge Management*, 1995.

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

[22] D. Djuric, "MDA-based Ontology Infrastructure", *Computer Science and Information Systems*, vol. 1(1), pp. 91–116, 2004.

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

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

[25] W3C – the World Wide Web Consortium, *The Semantic Web Initiative*, retrieved April 21, 2008 from http://www.w3.org/2001/sw, 2001.

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

[27] Q.H. Mahmoud, *Middleware for Communications: Concepts, Designs and Case Studies*. John Wiley and Sons, 2004.

[28] Q. Bing Y. Hongji, W.C. Chu, and B. Xu, "Bridging legacy systems to model driven architecture", in *Proceedings 27th Annual International Computer Software and Applications Conference COMPSAC 2003*, pp. 304-309, 2003

[29] F. Oquendo, "π-Method: a model-driven formal method for architecture-centric software engineering", *SIGSOFT Software Engineering Notes*, vol. 31(3), pp. 1-13, 2006.

[30] Y. Yang, X. Peng, and W. Zhao, "An Automatic Connector Generation Method for Dynamic Architecture", in *Proceedings International Computer Software and Applications Conference COMPSAC 2007*, pp. 409-414, 2007.

Dr. Claus Pahl is a Senior Lecturer at Dublin City University's School of Computing, where he is the leader of the Web and Software Engineering group. He has graduated from the Technical University of Braunschweig and has obtained a PhD from the University of Dortmund. He has published more than 160 papers in various journals, books, conference, and workshop proceedings. He is on the editorial board of four journals and is a regular reviewer for journals and conferences in the area of Web and Software technologies and their applications. He is the principal investigator of several basic and applied research projects in Web software engineering. His research interests cover a broad spectrum from service- and component technologies in software engineering to infrastructure and development technologies for Web applications such as e-learning.

Yaoling Zhu has recently finished his postgraduate research at the School of Computing at Dublin City University with an M.Sc. by Research. Yaoling is a graduate in Computer Science from the Zhengzhou Institute of Engineering, China. He has extensive experience in the software sector, working for several years as a senior software engineer for multinational companies such as Oracle, where he has been working on e-business outsourcing and Web service technologies in Oracle's European Development and Technology Centre. His research focuses on data integration problems in Web-based software systems.