Anti-Corruption Layer Modeling
Introducing a Model-Driven Approach to Integrate Legacy Software

Hendrik Bünder
buender@itemis.de
itemis AG

Abstract

Software development projects at an industrial scale, even greenfield projects, at some point require the integration of legacy systems. The anti-corruption layer pattern is often used to separate new and old system. However, implementing an anti-corruption layer is expensive and error-prone. The approach introduced by this paper creates a model of the legacy system by code analysis and model inference. Subsequently, the legacy model can be referenced by the new systems model and is thereby integrated into the model-driven development process. The paper shows how the anti-corruption layer is generated from the integrated models. Further, it will be reported on query support and first practical experience from an industrial scale project.

1 Introduction

Industry scaled software development projects at some point require the integration of legacy systems. The design of these systems has often rot over time leading to decreased extensibility and subsequently increased maintenance cost. Further, legacy systems often appear as a "big ball of mud" thereby making it difficult to analyze and reuse their artifacts.

As introduced by domain-driven design legacy systems can be integrated by implementing an anti-corruption layer [4]. The anti-corruption layer contains concepts from the legacy system that are required by the new systems. Thereby, existing services can be re-used without extending the legacy system or contaminating the new system with design flaws. Throughout the paper the focus is on reusing existing services and their data types. However, it also elaborates on how this information benefits the user interface. The approach introduced by this paper is based on the assumption that there is a model-driven software development process in place for modeling, generating, and implementing the new system [3].

In order to generate the anti-corruption layer with a model-driven development approach two questions must be answered. First, how to integrate legacy system artifacts into the new systems model? Second, which parts of the anti-corruption layer can be generated leveraging the model inherent information?

2 Building and Integrating a Model of the Legacy System

One key factor to success for modeling an anti-corruption layer is the preparation of the new systems meta model. First practical experience has shown that it is reasonable to mark all model elements representing the legacy system. Thereby, the model already expresses which elements are from the new or the old model, e.g. the introduced approach has a class "Service" and a class "Imported Service" from the new system and the legacy system, respectively.

Figure 1 illustrates a simplified meta model used for modeling and generating the anti-corruption layer. On the lower part of Figure 1 the artifacts created from the legacy system are defined. In the upper half the artifacts from the new systems model are presented. To avoid mapping the same attribute multiple times attribute definitions were separated from their usage in *data types* by introducing two concepts. First, in the new model a "Entity" contains all attributes and relations that logically belong together, e.g. all attributes of the entity customer or contract. The mapping is then specified between imported data types and the entities attributes. Second, we define *data types* that in the model do not hold own attributes, but rather reference attributes from entities. These data types are then used as input and output for services or as basis for user interface descriptions. By separating the attributes from their usage scenarios we only need to map attributes once and can leverage this information at multiple occasions.

Once the meta model is prepared the anti-corruption layer artifacts, such as imported services and their input and output data types, need to be instantiated. There are different ways of building a model of the legacy system. Obviously, it may be described manually, however, this approach is expensive and prone to error. Since legacy systems tend to be large and unclear about their internal dependencies, an automated approach is beneficial. There are multiple potential inputs for the process, such as existing UML [2] or Entity-Relationship models. The models can be transformed by a model-to-model transformation that instantiates the required artifacts in the anti-corruption layer section of the new systems

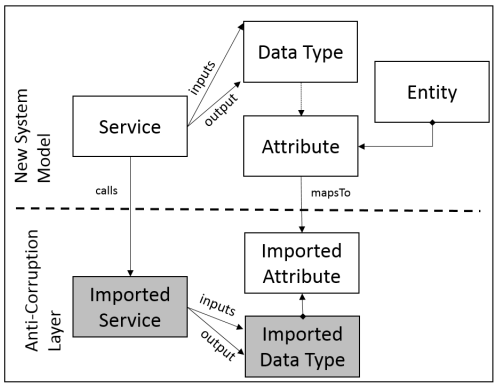

Figure 1: Anti-corruption Layer Modeling Meta Model

model. Yet, legacy system model and its source code may differ significantly. Therefore, it is more reasonable to analyze existing source code as first practical experience has shown. Based on the analysis a model of the legacy services and their input and output data types can be inferred [1].

After having instantiated the required legacy system model artifacts the usage from the new model, such as service calls can be modeled. In addition, an explicit data type mapping is required that specifies how the attributes map between new and old system. The modeled mapping information are basis for generating the anti-corruption layer and can be utilized to execute queries.

3 Leveraging the Model Integration

Based on the model artifacts and relations different parts of the anti-corruption layer are generated. Obviously, the specified "calls" reference enables the generation of service calls from new to old system. In addition, the model contains all information required to map input and output data types. Since the mapping is between imported data type and entity attributes, the definition of data types in the new model are decoupled from their mapping. In contrast, if a data type mapping per data type is required, developers will tend to model rather large new data structures to only specify one mapping. The introduced approach eliminates this restriction leading to mapped attributes being used in six different new data types on average. This strongly indicates, that the new data types are modeled based on their envisaged usage instead of the required effort to implement a mapping.

Applying this approach in real world projects revealed the problem of not-matching data formats. For example the legacy system might hold and transport date fields as String of eight characters while the new system handles dates as objects of class date. The introduced approach not only generates data type mapping, but also enables the generation of data format transformations, e.g. from String to date object.

Based on the data type mapping information a meta mapping, e.g. for exceptions and their messages, could be generated. In the legacy system exception messages may contain a reference to a specific field of the input data type. Thereby, the proprietary UI framework is able to highlight fields containing wrong or unexpected values. Since attributes might be named differently in the new system, the exception messages and their attribute references need to be mapped as well. As mentioned above the mapping is done between the imported type and the attributes of the entity which are then reused for different purposes, such as describing the UI. This implementation of the "Don't repeat yourself" principles allows to generate the meta mapping of the attribute references within the exception messages.

Since the model of the new system contains detailed information about the contained services, data types, and their relations queries and reports can be created. Leveraging the information inherent in the model to determine usage dependencies and impact analysis is basis for effectively and efficiently managing the integration of new and legacy system.

4 Conclusion

Introducing an anti-corruption layer to separate the new system from legacy components enforces separation of concerns and avoids contaminating the new system with design flaws. The paper has demonstrated the benefits of using a model-driven approach to automatically generate the anti-corruption layer implementation. It has been exemplified how separating the attribute mapping from the usage of these attributes in the new model decouples mapping from design decision. Finally, applying the approach in an industrial scale project has shown that modeling the anti-corruption layer decreases mapping effort, allows for generated data transformations, and enables advanced query and report functionality.

References

[1] Hendrik Bünder. "A UML-Agnostic Migration Approach From UML to DSL". In: *Softwaretechnik-Trends: Vol. 37, No. 2.* Berlin: Gesellschaft für Informatik e.V., Fachgruppe PARS, 2017, pp. 36–37.

[2] OMG. *UML - What is UML.* URL: `http://www.uml.org/`.

[3] Thomas Stahl. *Modellgetriebene Softwareentwicklung: Techniken, Engeneering, Management.* 2., aktual. und erw. Aufl. Heidelberg: Dpunkt.verlag, 2007. ISBN: 978-3898644488.

[4] Vaughn Vernon. *Domain-driven design distilled.* Boston: Addison-Wesley, 2016. ISBN: 978-0134434421.