Information and Software Technology 55 (2013) 357–364

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

System integration by developing adapters using a database abstraction

Arjan J. Mooij

Embedded Systems Institute, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 18 July 2011
Received in revised form 28 August 2012
Accepted 31 August 2012
Available online 8 September 2012

Keywords:
Model-driven development
Distributed systems
Adapter generation
Incremental view maintenance

ABSTRACT

Context: Large software systems are usually developed by integrating several smaller systems, which may have been developed independently. The integration of such systems often requires the development of a custom adapter (sometimes called mediator or glue logic) for bridging any technical incompatibilities between the systems.
Adapter development often focuses on how to respond to events from the external interfaces, e.g., by applying data conversions and performing events on (other) external interfaces. Such an operational focus is closely related to an implementation of the adapter, but it makes it complicated to reason about complex adapters. For example, it requires a lot of effort to understand the relation that the adapter establishes between the systems to be integrated, and to avoid any internal inconsistencies.
Objective: This article investigates a way to develop adapters in terms of a more abstract, model-based specification. Experts from the application domain should be able to reason about the specification, and the specification should contain enough details to generate an implementation.
Method: Based on a few industrial adapters from the domain of Maritime Safety and Security, we study ways to specify them conveniently, and ways to generate them efficiently. On this basis, we identify an approach for developing adapters. In turn, this approach is validated using an additional set of adapters.
Results: After abstracting from the details of the interface technologies, the studied adapters could be generated using techniques for incremental view maintenance. This requires an adapter specification in terms of database views to relate the main semantic concepts in the application domain.
Conclusion: For developing adapters, it can be useful to model all interface technologies as database operations. Thus adapters can be specified using database views, which improve the conceptual understanding of the adapters. Publish/subscribe-style adapters can then be generated using incremental view maintenance.

1. Introduction

Component-based software engineering (CBSE) and service-oriented computing (SOC) advocate the development of large software systems by integrating several smaller systems. In turn, these systems may have been developed independently, and hence there may be various technical incompatibilities between them [1]. These incompatibilities may range from the data that is communicated to the protocols that are used to communicate the data. An approach to resolve these incompatibilities, without changing the systems to be integrated, is to develop a custom adapter (sometimes called mediator or glue logic).

Adapter development often focuses on how to respond to events from the external interfaces, e.g., by applying data conversions and performing events on the (other) external interfaces. Such an operational focus is closely related to an implementation, but it makes it complicated to reason about complex adapters. For example, it requires a lot of effort to understand the relation that the adapter establishes between the systems to be integrated, and to avoid any internal inconsistencies.

The required functionality of an adapter is usually rather limited. This motivates the question whether it is possible to generate custom adapters automatically from some convenient (handmade) specification. This article investigates a way to develop adapters using an abstract, model-based specification. Experts from the application domain should be able to reason about the specification, and the specification should contain enough details to be able to generate an implementation.

We study a few industrial adapters from the domain of Maritime Safety and Security. Many of them interact with their environment through a DDS-based middleware, which uses a publish/subscribe architecture; a similar kind of interaction style is used for systems such as RSS, Twitter, etc. We identify a way to specify these industrial adapters conveniently, and to generate them efficiently. Afterwards, we validate this approach using an additional set of adapters from this application domain.

Contributions. We show that it can be useful to model all interfaces in terms of database operations. Thus we can apply all kinds

E-mail address: Arjan.Mooij@esi.nl

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.08.015

of techniques from the fields of databases and data mediation [2] to the development of adapters.

In particular, we show that the studied adapters can effectively be specified in terms of database views. Such a view-based specification is declarative in nature, and it focuses on *what* needs to be established instead of *how* it is established. An additional advantage is that, in general, data modelling is considered to be simpler (or better understood) than behavioural modelling.

We also show that, using such a database abstraction of the interfaces, publish/subscribe-style adapters can be generated using techniques for incremental view maintenance [3–6]. Thus, based on a declarative view-based adapter specification, we derive an operational adapter implementation in terms of how to respond to events on the external interfaces (compare also, e.g. [7]).

Overview. In Section 2 we discuss a motivating industrial case study, followed in Section 3 by our proposed adapter development approach. In Section 4 we illustrate this approach by providing additional technical details. In Section 5 we compare the adapter that we have developed with the proposed industrial prototype, and in Section 6 we discuss additional examples. Finally, in Section 7 we evaluate the approach, and in Section 8 we draw some conclusions.

2. Industrial case study

We consider an industrial prototype system for situational awareness from the domain of Maritime Safety and Security. This prototype system consists of several components for processing, integrating, and interpreting the vessel observations from several sensors (like radars). In particular, there are many adapters, which together cover a whole class of industrial adapters. As a running example we consider an adapter that integrates an additional sensor with the core system.

2.1. Core system

The components of the core system communicate with each other using a DDS-based middleware (Data Distribution Service for Real-time Systems [8]), which offers a data-centric publish/subscribe (DCPS) architecture for distributed systems. DDS-based middleware is used for mission-critical information management, also including automated financial trading and air traffic management. The PADRES platform [9] uses a similar paradigm in the context of distributed workflow management systems.

Components that interact with each other using a DDS do not directly address each other. Instead, they interact in terms of topics, which are named data channels that are stored in the DDS. Each topic is associated with a data type and a primary key; a topic can hold multiple instances of the data type if their values of the primary key are different. A basic interaction pattern is that one component publishes instances to a topic, and another component receives these instances by subscribing to the topic. Components can also query the instances of a topic (the last published instance per value of the primary key) in terms of SQL. Low-level formalizations of such primitive operations have been studied by [10–12]; we will adopt a more high-level perspective.

In such a publish/subscribe system, the most important artefact for integrating components is the data model. The fragment of the data model that we consider consists of three topics to represent the most-recent vessel information:

- *intel*: iid, mmsi, name;
- *system*: sid, iid, lid;
- *local*: lid, longitude, latitude, destination.

Each topic has a designated identifier field as the primary key, respectively, iid, sid, and lid. Topic *intel* represents static vessel data such as its name and its mmsi number, which is a maritime vessel identifier. Topic *local* represents dynamic vessel data such as its position (longitude and latitude) and (the name of) its destination. Topic *system* is primarily used to link instances of the other topics.

2.2. Additional sensor

The additional sensor is a maritime AIS (Automatic Identification System [13]) receiver. Every vessel has an on-board AIS transponder that uses several message types and reporting frequencies to broadcast information about the vessel. An AIS receiver collects these broadcasted AIS messages, and produces a TCP/IP stream of messages that are encoded using a standard from the NMEA (National Marine Electronic Association). We only consider the two main message types (after simplification) to represent the most-recent vessel information:

- *point*: mmsi, longitude, latitude;
- *voyage*: mmsi, destination, name.

Each message type contains the unique mmsi number of the reporting vessel. Message type *point* represents dynamic data such as position (longitude and latitude). Message type *voyage* represents relatively static data such as name and destination.

2.3. Integration

The interface of this additional sensor does not match the interface of the core system. The technical incompatibilities range from the data that is communicated to the protocols that are used to communicate the data. The incompatibilities between them can be classified using four criteria as described in Table 1; the extra column "AIS on DDS" is discussed afterwards.

The industrial prototype solution introduces an intermediate DDS-variant of AIS, as indicated in Table 1. Thus the integration is performed in two steps:

1. Adapter from AIS receiver to AIS on DDS: this adapter mainly deals with technicalities and low-level data conversions.
2. Adapter from AIS on DDS to the core system: this adapter deals with the main semantic differences in the domain concepts.

Both adapters were developed manually, and directly in terms of a general-purpose programming language. We have considered both adapters in isolation, and also their combination in a single adapter.

We focus on the adapter from AIS on DDS to the core system. This adapter achieves the following behaviour (details omitted). Once an AIS *point* message is received, all the three topics *intel*, *system*, and *local* are queried and afterwards updated or created. Once an AIS *voyage* message is received, the topic *intel* is queried and afterwards updated or created; the two topics *system* and *local* are also queried, and *local* is updated if already available.

Table 1
Case study: Overview of incompatibilities.

	AIS receiver	AIS on DDS	Core system
Protocol (concrete)	TCP/IP	DDS	DDS
Protocol (abstract)	Streaming	Publish/subscribe	Publish/subscribe
Data (abstract)	AIS messages	AIS messages	Custom model
Data (concrete)	NMEA	Records	Records