Mediation Patterns for Message Exchange Protocols

Stanislav Pokraev[1] and Manfred Reichert[2]

[1] Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
Stanislav.Pokraev@telin.nl, http://www.telin.nl/
[2] Center for Telematics and Information Technology,
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
M.U.Reichert@ewi.utwente.nl, http://www.ewi.utwente.nl/

Abstract. Systems interact with their environment (e.g., other systems) by exchanging messages in a particular order. Interoperability problems arise when systems do not understand each other's messages or follow incompatible message exchange protocols. In this paper we identify mismatches in message exchange protocols (involving two systems) and we propose solution patterns to compensate these mismatches.

1 Introduction

In order to interoperate systems must follow compatible message exchange protocols. For example, if one system first sends message M_1 and then message M_2, the partner system should be able to receive these two messages in the same order (i.e., M_1 before M_2). However, autonomous systems (especially when built in isolation) do not always use compatible message exchange protocols and therefore cannot interoperate. To compensate such mismatches and to make systems interoperable we need an additional system, which we denote as *process mediator*.

In this paper we identify the most common mismatches in message exchange protocols and propose respective process mediators to compensate these mismatches. Our findings are based on the result of a literature study [1][2][3][4] and a case study[5].

We illustrate the patters using the notation depicted in Figure 1.

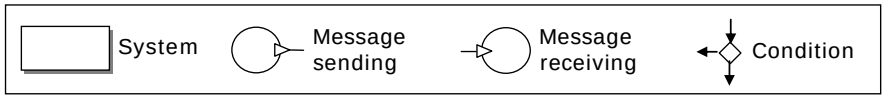

Figure 1. Notation

2 Mediation Patterns

In the following we summarize typical mismatches in message exchange protocols and propose mediation patterns to compensate them.

Problem: System A intends to send two messages, first M_1 and then M_2, whereas system B expects only message M_2.
Solution: Mediator M receives message M_1 and ignores it. Next, it receives message M_2 and forwards it to system B. This pattern is illustrated in Figure 2.

Problem: System B expects two messages, M_1 and M_2, whereas system A intends to send only message M_2.
Solution: Mediator M receives message M_2 from system A. Next, it uses *additional information* (either provided by another system or derived from the execution history) to construct and send message M_1 to system B. Finally, the mediator sends message M_2 to system B. Note, that this mismatch can only be compensated if mediator M has all information necessary to construct message M_1. This pattern is illustrated in Figure 3.

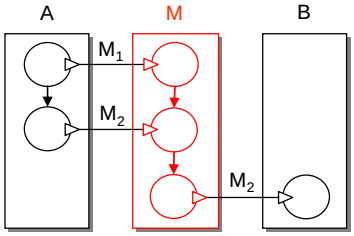

Figure 2. Unexpected message M_1

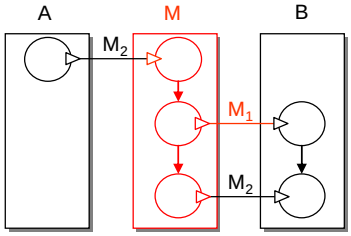

Figure 3. Insufficient message M_1

Problem: System A sends message M_1 to system B and continues without expecting an acknowledgement, whereas system B intends to send message M_{ack} to acknowledge the reception of message M_1.
Solution: Mediator M receives message M_1 from system A, sends it to system B, and then receives the acknowledgement M_{ack} on behalf of system A. This pattern is illustrated in Figure 4.

Problem: System A sends message M_1 and expects acknowledgement M_{ack} whereas system B does not intend to send such an acknowledgement.
Solution: Mediator M receives message M_1, sends it to system B, and then sends an acknowledgement (M_{ack}) to system A on behalf of system B. This pattern is illustrated in Figure 5.

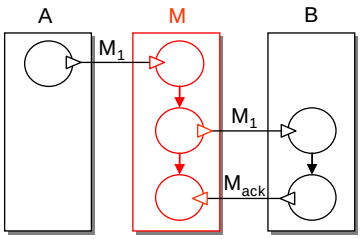

Figure 4. Unexpected acknowledgement

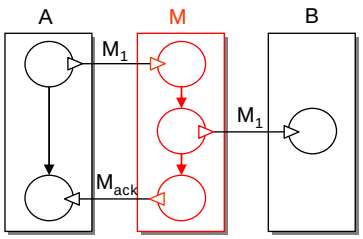

Figure 5. Insufficient acknowledgement

Problem: System A intends to send message M_1 first and then M_2, whereas system B expects first message M_2 and then M_1.

Solution: Mediator M receives first message M_1 and then message M_2. Next, it sends message M_2 first and then message M_1. This pattern is illustrated in Figure 6.

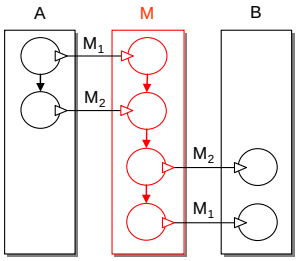

Figure 6. Message reordering

Problem: System B expects two messages M_2 and M_3 whereas system A intends to send only one message M_1 that contains both M_2 and M_3.

Solution: Mediator M first receives message M1. Then it uses the information from M1 to construct M2 and M3. Finally, the mediator sends M1 and M2 in the order expected by system B. This pattern is illustrated in Figure 7.

Problem: System B expects message M2 *n times* whereas system A intends to send only one message M1 that contains all ***n*** messages M2.

Solution: Mediator M first receives message M1. Then it starts a process of constructing M2 from the information in M1 and sending M2 to system B. This process is repeated until some condition evaluates to true. The pattern is illustrated in Figure 8.

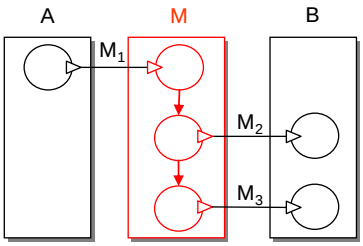

Figure 7. Message splitting

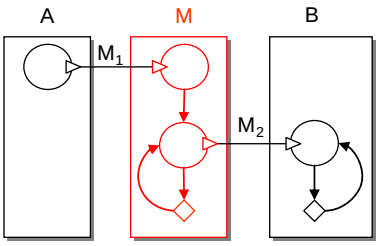

Figure 8. Message splitting

Problem: System A sends messages M1 and M2 whereas system B expects one message M3 that aggregates M1 and M2.

Solution: Mediator M receives both messages M1 and M2. Then it uses the information from these two messages to construct M3. Finally, the mediator sends M3 to system B. This pattern is illustrated in Figure 9.

Problem: System A sends message M1 *n* times whereas system B expects one single message M2 that aggregates all ***n*** messages M1.

Solution: Mediator M starts a process of receiving messages M1 until some condition evaluates to true. Next, it uses the information in the received messages to construct M2 and then sends M2 to system B. This pattern is illustrated in Figure 10.

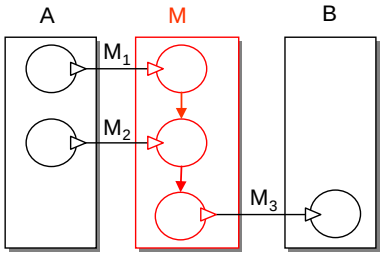

Figure 9. Message combining

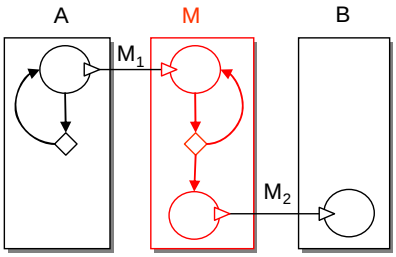

Figure 10. Message combining

3 Discussion

The presented mediation patterns address only mismatches in message exchange protocols. Some of the patterns (e.g., message splitting and aggregation) require *semantic mapping* between the data in the exchanged messages. Only if such mappings exist the mediators can construct an output message(s) provided input one(s).

More complex patterns can be constructed using the ones presented in the previous section. For example, splitting message M1 to three messages M2, M3 and M4 can be achieved by composing two ‘message splitting patterns’ (cf. Figure 11). Likewise,

changing the order of three messages can be achieved by extending the message reordering pattern (cf. Figure 12), etc.

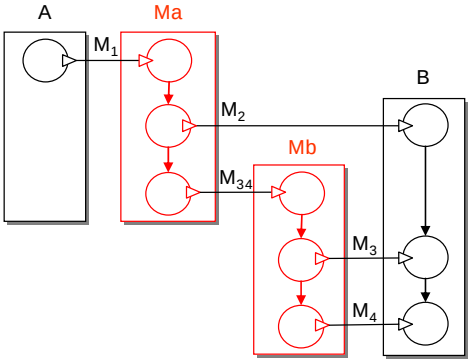

Figure 11. Composing message splitting patterns

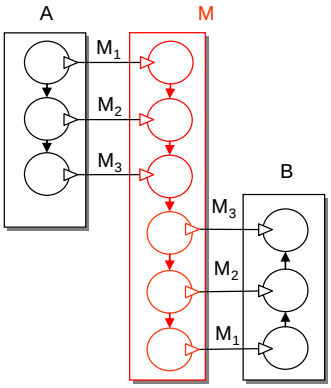

Figure 12. Extended message reordering pattern

4 Acknowledgments

The present work has been done in the Freeband Communication project A-Muse (http://a-muse.freeband.nl). Freeband Communication (http://www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025. The presented work a result of collaboration between the Telematica Instituut and the University of Twente, the Netherlands, which is partially supported by the Commission of the European Communities under the sixth framework programme (INTEROP Network of Excellence, Contract N° 508011, http://www.interop-noe.org/).

References

1. Bussler, C. B2B-Integration: Concepts and Architecture. Springer-Verlag, 2003.
2. Cimpian, E. and Mocan, A. Process Mediation in WSMX. WSMX Working Draft, 08 July 2005. http://www.wsmo.org/TR/d13/d13.7/v0.1/
3. Hohpe, G. and Woolf, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, 2004.
4. Pollock, J. T. and Hodgson, R. Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration. Wiley-Interscience, 2004.
5. Semantic Web Services Challenge 2006, http://sws-challenge.org