Mediating Connector Patterns for Components Interoperability

Romina Spalazzese, Paola Inverardi

Mediating Connector Patterns for Components Interoperability*

Romina Spalazzese and Paola Inverardi

Università degli Studi dell'Aquila
via Vetoio I-67100 L'Aquila, Italy
{romina.spalazzese,paola.inverardi}@di.univaq.it

Abstract. A key objective for ubiquitous environments is to enable system interoperability between system's components that are highly heterogeneous. In particular, the challenge is to embed in the system architecture the necessary support to cope with behavioral diversity in order to allow components to coordinate and communicate. In this paper we present the design building blocks for the dynamic and on-the-fly interoperability between heterogeneous components. Specifically, we describe an Architectural Pattern called *Mediating Connector*, that is the key enabler for communication. In addition, we present a set of *Basic Mediator* Patterns, that describe the basic mismatches which can occur when components try to interact, and their corresponding solutions.

Keywords: Heterogeneous Components Interoperability, Mediating Connector Architectural Pattern, Basic Mediator Patterns.

1 Introduction

A multitude of heterogeneous networked devices are today embedded in the Ubiquitous networked environment [2] where a key objective is to enable system interoperability. Tremendous work has been done in the middleware field while the application-layer interoperability remains an open problem calling for *mediating connectors* or *mediators*. The mediator concept was initially introduced to cope with the integration of heterogeneous data sources [13] and as design pattern [17]. In the field of software architectures ad hoc wrappers have been proposed to address communication problems [8]. Mediators and automated mediation have received attention within the Web Services and Semantic Web contexts [6,3,4]. Recently, the challenge is to provide general solutions to the behavioral diversities at runtime and on-the-fly, to respond to the continuous evolution of the environment[1]. An approach to protocol mediation is to categorize the types of protocol mismatches that may occur and that must be solved in order to provide corresponding solutions to these recurring problems. This immediately reminds of patterns [18,1,12,17] and of pattern-based works [15,16,20,21,22].

* The work is partly supported by the CONNECT European Project No 231167.

[1] CONNECT European project, http://connect-forever.eu/

M. Ali Babar and I. Gorton (Eds.): ECSA 2010, LNCS 6285, pp. 335–343, 2010.

In this paper we present a set of design building blocks for the interoperability between heterogeneous components which would certainly facilitate the solution. The contributions of this paper are: (1) an Architectural Pattern called *Mediating Connector*, that is the key enabler for communication; (2) a set of *Basic Mediator* Patterns that describe: (i) the basic mismatches which can occur while components try to interact, and (ii) their corresponding solutions. The paper is organized as follows: in Section 2, we sketch a pattern-based approach for the automatic synthesis of Mediating Connector. In Section 3, we illustrate the Mediating Connector Architectural Pattern for the ubiquitous networked environment. In Section 4, we show the Basic Mediator Patterns and we conclude, in Section 5, by outlining future work.

2 A Pattern-Based Approach for Interoperability Mismatches

In this section we describe our proposal for an automated pattern based approach, whose details are in [9]. For the sake of this paper, we make some assumptions and we investigate the related underling research problems as part of CONNECT. We assume to know the interaction protocols run by two networked components as Labeled Transition Systems (LTS) [14] and the components' interfaces with which to interact as advertised or as result of learning techniques [5,19]. We also assume a semantic correspondence between the messages exchanged among components exploiting ontologies. The first step is to establish whether the components are *potentially compatible*, i.e., if it makes sense for them to *interoperate* through the Mediating Connector. This amounts to understand if the components share some *intent* (trace), i.e., if they have some complementary sequences of messages visible at interface level. To do this we define (1) a decomposition strategy/tool to decompose the whole components' behavior (LTS) into *elementary behaviors* (traces) representing *elementary intents* of the components and then (2) an automatic analyzer to identify mismatches between elementary behaviors of the different components. Once discovered the components compatibility, solving their interoperability means solving the behavioral mismatches that they exhibit. Then it is necessary to: (3) define a mismatches manager to solve the identified mismatches between elementary behaviors; (4) define a composition approach to build elementary mediating behaviors (mediating traces) based on the identified mismatches and their relative solutions; (5) define a composition strategy to build a mediating connector's behavior starting from the elementary mediating behaviors.

The above described approach is far from trivial, especially to achieve automatically. However, in the following we show its feasibility. To address steps (1) and (5) the approach makes use of a compositional strategy to decompose components interaction protocols into traces and compose mediating connectors interaction protocol from mediating traces respectively. Furthermore, we describe six Basic Mediator Patterns that are the building blocks on which the steps (2), (3), and (4) can be built upon.

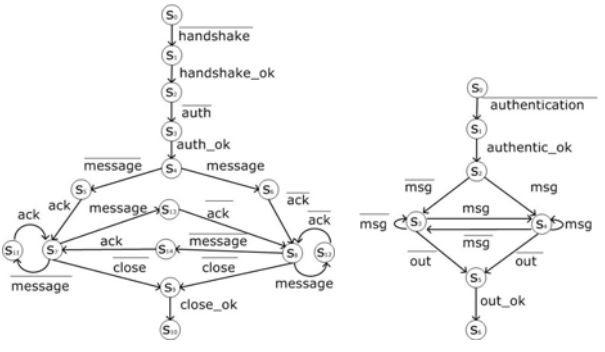
3 Mediating Connector Architectural Pattern

We characterize the interoperability problem between diverse components populating the ubiquitous environment and its related solution as a *Mediating Connector Architectural Pattern* basing on the template used in [1]. The Mediating Connector is a behavioral pattern. Being an architectural building block embedding the necessary support, it should be used to dynamically cope with components' behavioral diversity.

Name. Mediating Connector.

Also Known As. Mediator.

Example. We describe the example used in [10] where we have been studying the problem and where appeared first results on the theory underlying our approach. An extended and more complete version of the theory can be found in [9]. We consider the simple yet challenging example of instant messaging. Various instant messaging systems are now in use. However, although those systems implement similar functionalities, end-users need to use the very same system to communicate due to behavioral mismatches of the respective protocols. For instance, consider Windows Messenger[2](WM), now called Windows Live Messenger, and Jabber Messenger[3](JM). Figure 1 models their behavioral protocols using LTSs. We use the usual convention that actions with overbar denote output actions while the ones with no overbar denote input actions. These systems should be able to interoperate since they both amount to supporting authentication with their servers and then message exchanges among peers. However mediating their respective protocols to achieve interoperability is far from trivial, especially if one wants to achieve a general solution. An effort has been done in [11] requiring the implementation of the translation from any client protocol to a reference exchange protocol, and vice versa. Unfortunately this affects the generality and the automation of the approach.



(a) Windows protocol (b) Jabber protocol

Fig. 1. Behavioral models of two instant messengers

Context. The environment is distributed and changes continuously. Heterogeneous mismatching systems require seamless coordination and communication.

Problem. In order to support existing and future systems' interoperability, some means of mediation is required. From the components' perspective, there

[2] Windows Live Messenger, http://www.messenger.it/

[3] Jabber Software Foundations, http://www.jabber.org/

should be no difference whether interacting with a peer component, i.e, using the very same interaction protocol, or interacting through a mediator with another component that uses a different interaction protocol. The component should not need to know anything about the protocol of the other one while continuing to "speak" its own protocol. Using the Mediating Connector, the following *forces* (aspects of the problem that should be considered when solving it [1]) need to be balanced: (a) the different components should continue to use their own interaction protocols. That is, components should interact as if the Mediating Connector were transparent; (b) the following *basic interaction protocol mismatches* should be solved in order for a set of components to coordinate and communicate (cfr. Section 4): 1) Extra Send/Missing Receive Mismatch; 2) Missing Send/Extra Receive Mismatch; 3) Signature Mismatch; 4) Ordering Mismatch; 5) One Send-Many Receive/Many Receive-One Send Mismatch; 6) Many Send-One Receive/One Receive-Many Send Mismatch.

Solution. The introduction of a Mediating Connector to manage the interaction behavioral differences between potentially compatible components. The idea behind this pattern is that components that would need some interaction protocol's adaptation to become compatible, and hence to interoperate, are able to coordinate and communicate achieving their goals/intents without undergoing any modification. The Mediating Connector is one (or a set of) component(s) that manage the behavioral mismatches. It directly communicates with each component by using the component's proper protocol. The mediator forwards the interaction messages from one component to the other by making opportune translation/adaptation of protocols when/if needed.

Structure. The Mediating Connector Pattern comprises three types of participating components: communicating components, potentially compatible components and mediators. Figure 2 shows the objects involved in a mediated system. The *communicating components* implement already compatible components, i.e. able to interact and evolve following their usual interaction behavior. The *potentially compatible components* implement application level entities that want to reach some of their intents by interacting with other components able to satisfy their needs, i.e. required/provided functionalities. However those components are unable to directly interact because of protocol mismatches and can only evolve following their usual interaction behavior, without any change. The *mediators* are entities responsible for the mediated communication between the components. This means that the role of the mediator is to make compatible components that are mismatching. That is, a mediator must receive and properly forward requests and responses between potentially compatible components.

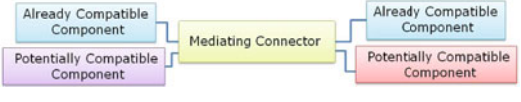


Fig. 2. Entities involved in a mediated system

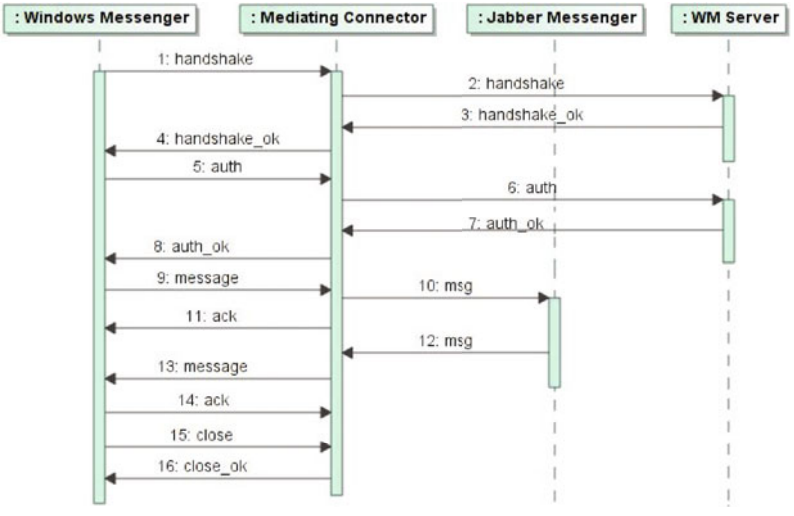


Fig. 3. Scenario on the relevant operation of a Mediator

Dynamics. Figure 3 illustrates the interactions between three components and one mediator belonging to the messengers example. Triggered by a user, the Windows Messenger protocol (Figure 1(a)) performs one of its possible behavior: it authenticates after an handshake, sends/receives several messages, and closes. The mediator should: (1) forward the handshake and authentication messages as they are between the Windows Messenger and its authentication server (communicating components), (2) translate and forward messages between the Windows and Jabber Messengers (potentially compatible components)[4] (3) forward the closing messages as they are between the Windows Messenger and its server (communicating components).

Implementation. The implementation of this pattern implies the definition of an approach/tool (we have proposed one in Section 2) to automatically synthesize the behavior of the Mediating Connector which allows potentially compatible components to interoperate mediating their interactions.

Example Resolved. The Mediating Connector's concrete protocol for the example is shown in Figure 4. Once established that the two messengers are potentially compatible, i.e., they have some complementary portion of interaction protocols, the mediating connector manages the components' behavioral mismatches allowing them to have a mediated coordination and communication.

Variants. Distributed Mediating Connector. It is possible to implement this pattern either as a centralized component or as distributed components. This latter introduces a synchronization issue that has to be taken into account.

[4] With "translation" we refer to a "behavioral translation" (see Section 4).

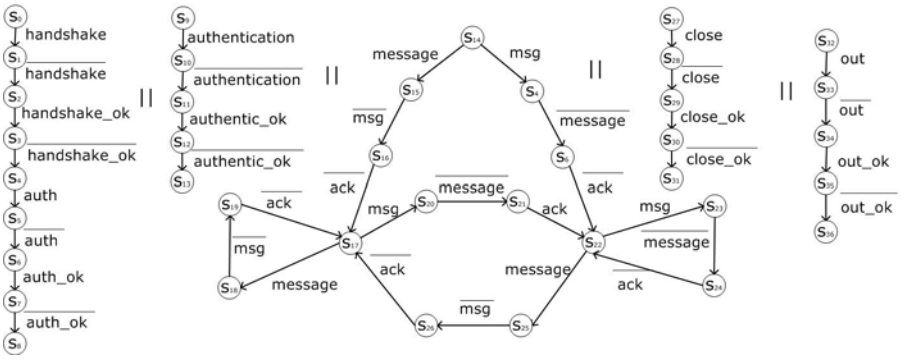


Fig. 4. Mediating Connector protocol of the messengers example

Consequences. The main *benefit* of the Mediating Connector Pattern is that it allows interoperability between components that otherwise would not be able to do it because of their behavioral differences.

The main *liability* that the Mediating Connector Pattern imposes is that the systems using it are slower than the ones interacting directly because of the indirection layer introduced. However the severity of this drawback is mitigate and made acceptable by the fact that such systems, without mediator, are not able at all to interoperate.

4 Basic Mediator Patterns

The *Basic Mediator Patterns* include the above mentioned basic interoperability mismatches together with their corresponding solutions. These patterns are: (1) Message Consumer Pattern, (2) Message Producer Pattern, (3) Message Translator Pattern, (4) Messages Ordering Pattern, (5) Message Splitting Pattern, (6) Messages Merger Pattern.

Figure 5 shows, for each Basic Mediator Pattern: (i) two traces (left hand-side and right hand-side), showing the basic interoperability mismatch coming from two potentially compatible components, and (ii) its related basic solution trace (in the center). All the considered traces are the most elementary in terms of messages exchanged and only their visible messages are shown. The mismatches, inspired by service composition mismatches, refer to send/receive problems that can occur while synchronizing two traces. In real cases, the traces may also contain portions of behavior already compatible (abstracted by dots in the figure) and may amount to any combination of the presented mismatches. Then an appropriate strategy to detect and manage this is needed. The basic patterns, share the context considering two traces (left and right) expressing similar complementary functionalities and focusing on one of their semantically

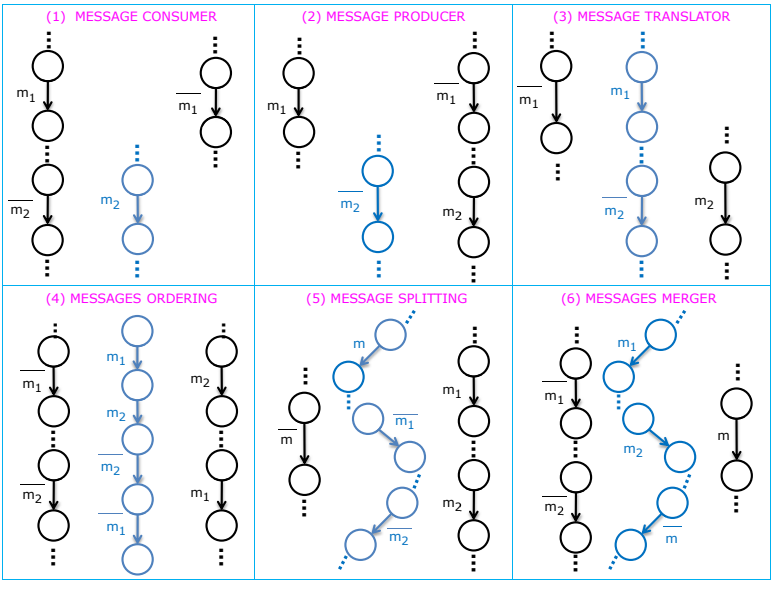


Fig. 5. Basic Mediator Patterns

equivalent elementary actions. Moreover the patterns have the same intent: to allow synchronization between the two traces letting them evolve together which otherwise would not be possible because of behavioral mismatches. Due to the lack of space, we do not give here further details that can be found in [7].

5 Conclusion

The Ubiquitous environment, embedding a big number of heterogeneous system's components, puts forward an ever growing need of mediation entities for component's interoperability purpose. The challenge is to embed *mediators* components into the system architecture allowing mismatching components to coordinate and communicate. Another challenge is to find dynamic and on the fly approaches to cope with component's behavioral diversities. To respond to these two challenges, we illustrated the Mediating Connector Architectural Pattern which, encapsulating the necessary support, is the key enabler for the communication between mismatching components. We also proposed an automatic pattern based approach describing a set of Basic Mediator Patterns, including basic mismatches and respective solutions, which represent the basic building blocks on which an automatic approach can build upon.

As future works, we intend to define a theoretical compositional strategy to allow reasoning on mismatches and to build the mediating connector behavior. Moreover we also aim at providing the "concrete" Basic Mediator Patterns, i.e.,

the skeleton code corresponding to the "abstract" ones presented in this work and present the actual code for the component's behavior decomposition and the mediating connector behavior building.

References

1. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns, vol. 1. John Wiley & Sons, Chichester (August 1996)
2. Weiser, M.: The computer for the 21st century. Scientific American (September 1991)
3. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007, pp. 993–1002. ACM, New York (2007)
4. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol mediation for adaptation in semantic web services. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 635–649. Springer, Heidelberg (2006)
5. Issarny, V., Steffen, B., Jonsson, B., Blair, G., Grace, P., Kwiatkowska, M., Calinescu, R., Inverardi, P., Tivoli, M., Bertolino, A., Sabetta, A.: CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In: ICECCS 2009, pp. 154–161 (2009)
6. Vaculin, R., Neruda, R., Sycara, K.P.: An Agent for Asymmetric Process Mediation in Open Environments. In: Kowalczyk, R., Huhns, M.N., Klusch, M., Maamar, Z., Vo, Q.B. (eds.) SOCASE 2008. LNCS, vol. 5006, pp. 104–117. Springer, Heidelberg (2008)
7. Spalazzese, R., Inverardi, P.: Mediating Connector Patterns for Components Interoperability. Tech. Rep., University of L'Aquila (2010)
8. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE 2003, pp. 374–384. IEEE Computer Society, Washington (2003)
9. Spalazzese, R., Inverardi, P., Issarny, V.: A Theory of Mediators for the Ubiquitous Networking Environment - Version 2. Tech. Rep. TRCS 002/2010 (2010)
10. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: WICSA/ECSA 2009, pp. 345–348 (2009)
11. Motoyama, M.A., Varghese, G.: Crosstalk: scalably interconnecting instant messaging networks. In: WOSN 2009, pp. 61–68. ACM, New York (2009)
12. Avgeriou, P., Zdun, U.: Architectural Patterns Revisited – A Pattern Language. In: EuroPLoP 2005, Irsee, Germany, 139 Pages (2005)
13. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25, 38–49 (1992)
14. Keller, R.M.: Formal verification of parallel programs. Commun. ACM 19(7), 371–384 (1976)
15. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
16. Cimpian, E., Mocan, A.: Wsmx process mediation based on choreographies. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 130–143. Springer, Heidelberg (2006)

17. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Resusable Object-Oriented Software. Addison-Wesley Professional, Reading (1995)
18. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Center for Environmental Structure Series, vol. 2. Oxford University Press, New York (1977)
19. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proc.ESEC/FSE, pp. 141–150 (2009)
20. Li, X., Fan, Y., Jiang, F.: A classification of service composition mismatches to support service mediation. In: GCC, pp. 315–321 (2007)
21. Li, X., Fan, Y., Wang, J., Wang, L., Jiang, F.: A pattern-based approach to development of service mediators for protocol mediation. In: WICSA 2008, pp. 137–146. IEEE Computer Society, Los Alamitos (2008)
22. Jiang, F., Fan, Y., Zhang, X.: Rule-based automatic generation of mediator patterns for service composition mismatches. In: Proc. of GPC-WORKSHOPS 2008, pp. 3–8. IEEE Computer Society, Washington (2008)