Revisiting the Middleware Paradigm: On-the-fly Interoperability in Highly Complex Distributed Systems

Amel Bennaceur, Gordon Blair, Nikolaos Georgantas, Paul Grace, Paola Inverardi, Valérie Issarny, Animesh Pathak, Rachid Saadi, Romina Spalazzese

Revisiting the Middleware Paradigm: On-the-fly Interoperability in Highly Complex Distributed Systems

Amel Bennaceur[1], Gordon Blair[2], Nikolaos Georgantas[1], Paul Grace[2], Paola Inverardi[3], Valérie Issarny[1], Animesh Pathak[1], Rachid Saadi[1], and Romina Spalazzese[3]

[1] INRIA, CRI Paris-Rocquencourt, France
[2] Lancaster University, UK
[3] University of L'Aquila, Italy http://connect-forever.eu/

Abstract. Distributed systems are becoming increasingly complex. We are moving from a world where we provide domain-specific middleware platforms (e.g., for Enterprise systems, Grid, MANET, ubiquitous environments) to one where these technology-dependent islands are themselves dynamically composed and connected together to create richer, dynamically deployed systems. Existing middleware approaches and paradigms are simply unable to cope with the demands of such heterogeneous and dynamic environments. Indeed, as we move towards a world of systems of systems, we can say that middleware is in crisis, being unable to deliver on its most central promises, which is offering interoperability, i.e., the ability of one or more systems to connect, understand and exchange data with one another. We argue that this requires a fundamental re-think of the architectural principles and techniques underpinning middleware platforms. We need to turn from relatively static solutions based on promoting a particular interoperability solution or bridging strategy, to much more dynamic solutions where we generate appropriate machinery for interoperability on the fly. This promotes an approach that may be termed emergent middleware designed to solve interoperability at runtime according to what is discovered and needed in a given context. This paper reports on the results of the CONNECT project, a collaborative initiative bringing together experts in middleware and software engineering, semantic modeling of services, and formal foundations of distributed systems, which together provide key building blocks for enabling emergent middleware.

Keywords: Interoperability, Middleware, Ontology, Pervasive networking, Protocol mediation, Semantic matching

1 Introduction

Distributed systems are becoming increasingly complex. We are moving from a world where we provide domain-specific middleware platforms (e.g., for Enterprise systems, Grid, MANET, ubiquitous environments) to one where these

technology-dependent islands are themselves dynamically composed and connected together to create richer, dynamically deployed systems. While there are many challenges to obtaining this overall goal, the most fundamental one is '*interoperability*', i.e., the ability of one or more systems to connect, understand and exchange data with one another, which is challenged by:

- *Extreme heterogeneity.* Pervasive sensors, embedded devices, PCs, mobile phones, and supercomputers are connected using a range of networking solutions, network protocols, middleware protocols, application protocols and data types. Each of these can be seen to add to the plethora of technology islands (i.e., systems that cannot interoperate).
- *Dynamic/spontaneous Communication.* Connections between systems are not made until runtime; no design or deployment decision, e.g., choice of middleware, can inform the interoperability solution.

Existing middleware approaches and paradigms are simply unable to cope with the demands of such heterogeneous and dynamic environments. Indeed, the trivial approach that is to adopt a given middleware solution throughout the complex system is limited for a number of reasons: i) the chosen solution may not work well in all operating environments (for example in MANET environments), ii) a given middleware solution today is a legacy system of the future and hence this solution has a limited longevity, and iii) middleware itself adds to the interoperability problem especially given the plethora of solutions available today combined with the lack of interoperability across these sets of platforms. Other solutions such as bridging are also unsuitable for highly heterogeneous and dynamic systems as we explain below. Furthermore, the openness and dynamics of today's pervasive networks require us to deal with higher level heterogeneity (above the middleware layer), i.e., the heterogeneity of data and application semantics. This problem has been well studied in the Semantic Web community but this work assumes a given approach to interoperability and also is not well integrated into contemporary middleware practice.

It is clearly the role of middleware to respond to these demands of extreme heterogeneity and dynamism. Middleware must *mediate* the interactions in complex distributed systems where entities dynamically meet and *functionally match* without prior knowledge of each other. Such mediators must preserve the original semantics of legacy components, while adapting the components' behavior for them to effectively interact. However, although the mediator paradigm has been studied quite extensively (spanning data, protocol, and functional levels), we flag the need for *on-the-fly mediation* leading towards universal interoperability in complex distributed systems. This remains a significant challenge. In other words, a fundamental re-think of the middleware paradigm is needed so that middleware solutions *emerge* from the dynamic encounter of networked systems. This is such a challenge that we are investigating as part of the European long-term research project CONNECT (`http://connect-forever.eu/`) that gathers experts in middleware and software engineering, semantic modeling of services, and formal foundations of distributed systems. The goals of this paper are as follows:

- To highlight the potential crisis in distributed systems in terms of achieving the basic property of interoperability in complex distributed systems and to stimulate debate around this crucial issue;
- To ground this perspective in a study of the state of the art of middleware and related interoperability solutions;
- To highlight the need for on-the-fly interoperability and emergent middleware as promoted by the CONNECT project;
- To present initial thoughts resulting from the CONNECT project in terms of the theoretical foundations of on-the-fly mediation and further practical application.

The next section briefly surveys state-of-the-art interoperability solutions, highlighting the rich and diverse baseline but yet the need to go beyond the traditional middleware paradigm. Section 3 then formalizes the foundations of the proposed approach to on-the-fly interoperability, which relies on automated reasoning about mediated matching between ontology-based middleware-agnostic models of interaction protocols. Section 4 focuses on the application of the theoretical foundations, discussing the actual realization of mediated protocol matching using a concrete case study. Finally, Section 5 concludes with a summary of our contribution.

2 State-of-the-Art Interoperability Solutions

Achieving interoperability between independently developed systems has been one of the fundamental goals of middleware researchers and developers; and prior efforts have largely concentrated on solutions where conformance to one or other standard is required, e.g., as illustrated by the significant standards work produced by the OMG for CORBA middleware (`http://www.corba.org/`), and by the W3C for Web Services based middleware (`http://www.w3.org/2002/ws/`). Such solutions have been successful in connecting systems developed on different hardware platforms, operating systems and in different programming languages. Indeed, if a single standard for the development of distributed systems had been agreed upon, the problem of interoperability would have been solved. Unfortunately, this is not the case due to the heterogeneity of middleware solutions themselves. While applications developed upon the same platform can interoperate with one another, applications developed on different middleware platforms cannot. For example, protocols such as SOAP and Java RMI cannot interoperate; and all RMI protocols cannot interoperate with all tuple-space or message-based protocols. Further, as systems are independently developed, the semantics of their data and functionalities rarely match. Hence, to overcome such heterogeneity, the following five important interoperability dimensions must be considered, while, as discussed next, the state of the art tackles only a few of them:

1. *Discovery protocol interoperability:* It should be possible for networked systems to advertise to, and find one another irrespective of the discovery protocol they themselves employ.

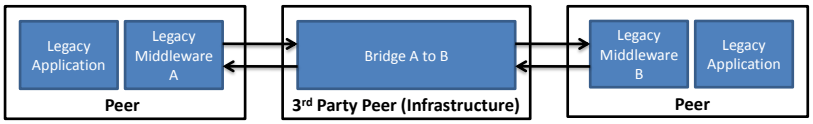


Fig. 1. Software bridges

2. *Interaction protocol interoperability:* Two or more networked systems whose interaction protocols (e.g., RMI, messaging, etc.) differ can be bound together in order to interoperate.
3. *Data interoperability:* The application data of the networked systems must be semantically equivalent between the two parties, and transformed to a format that the receiver can understand and process (after the binding between the heterogeneous protocols has been created).
4. *Application interoperability:* Differences between application interfaces can be resolved.
5. *Interoperability of non-functional properties.* Interoperability can be achieved between systems while maintaining the non-functional properties of each.

2.1 Bridging

Software bridges enable peers in one middleware domain to interoperate with peers in another. The bridge acts as a one-to-one mapping between domains; it will take messages from a peer in one format and then marshal this to the format of the peer middleware; the response is then mapped to the original message format (see Fig. 1). Note, appropriate discovery and naming services are used to ensure that an application is pointed to the bridge (rather than the original service endpoint); hence, additional middleware deployment is required to complete the interoperability solution (the bridge alone is insufficient). Many bridging solutions have been produced between established commercial platforms, e.g., [27, 18]. While a recognized solution to interoperability, bridging is infeasible in the long term as the number of middleware systems grow, i.e., due to the effort required to build direct bridges between all of them.

Enterprise Service Buses (ESB) can be seen as a special type of software bridge; they specify a service-oriented middleware with a message-oriented abstraction layer atop different messaging protocols (e.g., SOAP, JMS, SMTP). Rather than provide a direct one-to-one mapping between two messaging protocols, a service bus offers an intermediary message bus. Each service (e.g., JMS queue, Web Service etc.) maps its own message onto the bus. The bus then transmits the intermediary messages to the corresponding endpoints that reverse the translation to the local message type. Hence traditional bridges offer a 1-1 mapping; ESBs offer an N-1-M mapping.

Bridging solutions have shown techniques whereby two protocols (discovery or interaction) can be mapped onto one another. These can either use a one-to-one mapping or an intermediary bridge; the latter allowing a range of protocols

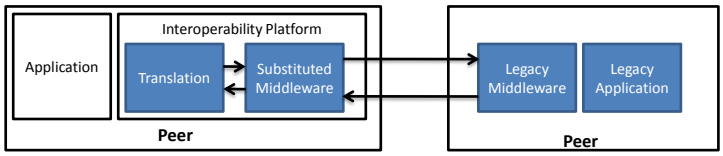


Fig. 2. Interoperability platform

to easily bridge between one another. This is one of the fundamental techniques to achieve interoperability; however, the technologies themselves are limited in the dimensions they consider since only discovery and interaction protocol interoperability are tackled. Furthermore, the bridge is usually a known element that each of the end systems must be aware of and connect to in advance; this limits the potential for two legacy-based applications to interoperate.

2.2 Interoperability Platforms

Figure 2 illustrates the key elements of approaches that provide an interoperability platform for client or peer applications to be implemented directly upon. For example, for a client-side application, it guarantees that the application can interoperate with all networked systems irrespective of the middleware technologies they employ. First, the interoperability platform presents an API for developing applications with. Secondly, it provides a substitution mechanism where the implementation of the protocol to be translated to, is deployed locally by the middleware to allow communication directly with the legacy peers (which are simply legacy applications and their middleware). Thirdly, the API calls are translated to the substituted middleware protocol. A key feature of this approach is that it does not require reliance on interoperability software located elsewhere, e.g., a remote bridge or an infrastructure server; this makes it ideal for infrastructureless environments. A number of interoperability platforms have been proposed in the literature, including: UIC [30], ReMMoC [17], and WSIF [11].

For the particular use case where you want a client application to interoperate with everyone else, interoperability platforms are a powerful approach that has demonstrated this is achievable. However, these solutions rely upon a design-time choice to develop applications upon the interoperability platforms. Therefore, they are unsuited to other interoperability cases, e.g., when two applications developed upon different legacy middleware want to interoperate spontaneously at runtime. Within this scope, discovery and interaction protocols interoperability have been addressed, while the other three dimensions have not been explicitly considered.

2.3 Transparent Interoperability

In the interoperability platform approach, at least one endpoint is aware of the interoperability problem and employs a framework to resolve it. In transparent

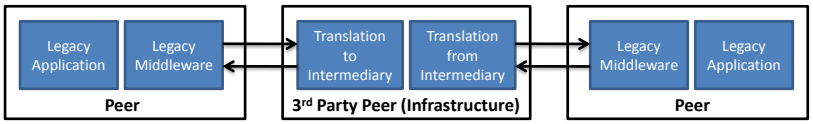


Fig. 3. Transparent interoperability

interoperability solutions, neither application is aware, and hence legacy applications can be made to communicate with one another. Figure 3 shows the key elements of the approach. Here, the protocol-specific messages, behavior and data are captured by the interoperability framework and then translated to an intermediary representation (note the special case of a one-to-one mapping, or bridge, is where the intermediary is the corresponding protocol); a subsequent mapper then translates from the intermediary to the specific legacy middleware to interoperate with. The use of an intermediary means that one middleware can be mapped to any other by developing these two elements only (i.e., a direct mapping to every other protocol is not required). Another difference to bridging is that the peers are unaware of the translators (and no software is required to connect to them, as opposed to connecting applications to 'bridges').

There is a number of variations of this approach, in particular where the two parts of the translation process are deployed. They could be deployed separately or together on one or more of the peers (but in separate processes that are transparent to the application) although they are commonly deployed across one or more infrastructure servers. Existing solutions include INDISS [5], uMiddle [25], OSDA [20] and SeDiM [13]. However, the technologies are limited in the dimensions of interoperability they consider, ignoring application interoperability.

2.4 Semantic-based Interoperability

The interoperation solutions proposed above concentrate on the middleware layer. But, by and large they ignore the data and application dimensions. For two parties to interoperate, it is not enough to guarantee that the data flow across; they must both build a semantic representation of the data that is consistent across the components boundaries. Historically, the problem has been well known in the database community where there is often the need to access information on different databases that do not share the same data schema. This has resulted in the extensive study of database federation and mediation services [38]. More recently, with the advent of open architectures, such as Web Services (WS), the problem is to guarantee interoperability at all levels [33]. Then, it is no surprise that significant work on interoperability has been ongoing in the area of Semantic WS, specifically targeting mediation among networked services. In particular, the WSMO effort (`http://www.wsmo.org/`) stresses the representation of goals and mediators as 'first class citizens' and defines three levels of mediation: data, protocol and business-process.

The original motivation for semantics in conjunction with WS has been to support automatic service discovery and composition [23]. Related concern is that of solving possible protocol mismatches among composed services. This is also known as the problem of protocol conversion/mediation, which is computationally hard for protocols specified as finite state systems. Still, tractable solutions are possible for some classes of systems by placing restriction on protocols [4, 16, 6, 35], while other approaches focus on overcoming signature/data type mismatches [39, 19]. Various solutions to protocol conversion targeted at WS have then emerged since the mid 2000s; they aim at (semi-)automatic design-time adaptation of interaction protocols with networked services so as to ease service composition [10, 33, 26, 37, 34] and/or substitution [29] in business processes. However, work on protocol mediation concentrates on application-layer protocols (business processes), assuming homogeneous middleware. In addition, very few work considers fully automated on-the-fly adaptation of business processes, with the notable exception of [8].

Complementary to the above, a number of research efforts have investigated middleware that support semantic specification of services. These solutions mainly focus on providing middleware functionalities enabling semantic service discovery and composition [22, 31, 9, 1]. Overall, the semantic-based interoperability approaches address the key data and application level heterogeneity problems; however, they assume development upon a common service-oriented middleware.

2.5 Summary

The results of our state-of-the-art investigation shows two important things: (1) there is a clear disconnect between the main stream middleware work (§ 2.1 to 2.3) and the work on application, data, and semantic interoperability (§ 2.4); (2) none of the current solutions addresses jointly all of the interoperability challenges.

With respect to the first problem, it is clear that two different communities evolved independently. The first one, addressing the problems of middleware, has made a great deal of progress toward middleware that support sophisticated discovery and interaction between services and components. The second one, addressing the problem of semantic interoperability between services; however, inflexibly assuming WS as the underlying middleware. The research work on semantic middleware shows that ultimately the two communities are coming together, but a great deal of work is still required to merge the richness of the work performed on both sides.

With respect to the second problem, no solution attempts to resolve all five interoperability dimensions. Those that concentrate on application and data, e.g., Semantic WS rely upon a common standard (WSDL). Further, all solutions require conformance (at some level) to a particular abstraction to achieve interoperability. For example, IDL and WSDL are visible abstractions that applications are developed with. In transparent solutions, domain-specific abstractions

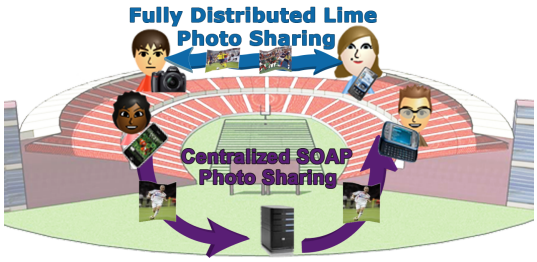


Fig. 4. Pervasive photo sharing

underpin the intermediary representation. Hence no solution covers the full diversity of communication abstractions, e.g., making tuple spaces, message-based, RPC and publish-subscribe systems interoperate in a spontaneous, transparent fashion.

The next section sets the theoretical foundations of our approach to on-the-fly interoperability, i.e., *emergent middleware*, which provides a framework to address the five interoperability challenges. However, in this paper, we concentrate on the issue of on-the-fly interaction and application interoperability (i.e., supporting interoperability among networked systems from the application down to middleware layers), while dealing with the non-functional dimension is part of our future work. Regarding discovery protocol and data interoperability, we exploit significant background in the area, i.e., interoperable service discovery [2, 7] and ontology [12]. In addition, in the following, without loss of generality, we focus on interoperability between two interacting networked systems.

3 Theoretical Foundations of Emergent Middleware

To illustrate the need for emergent middleware, we consider the simple yet challenging scenario of photo sharing in a stadium (Fig. 4), which enables spectators to exchange the pictures of the most significant happenings from their perspective (e.g., goals in the case of football games). Different implementations of *Photo-Sharing* may be envisioned, from centralized to fully distributed. In the centralized implementation, the stadium offers the *Photo-Sharing* service and the spectators' smartphones run service clients to upload and download pictures; typical supporting middleware solution is RPC-based using, e.g., WS middleware. In the distributed implementation, the spectators' smartphones run a peer-to-peer application for photo exchanges, for which distributed shared memory *à la* tuple space is a middleware of choice.

Figure 5 depicts the interaction protocols run by the networked systems involved in the scenario for both the (a) shared memory and (b) RPC implementations. Protocols are depicted as state machines representing the behaviors of systems in terms of their interactions with other systems, and the transitions are labeled by the corresponding networked systems' actions. An action is specifically characterized by the associated middleware and application functions denoted

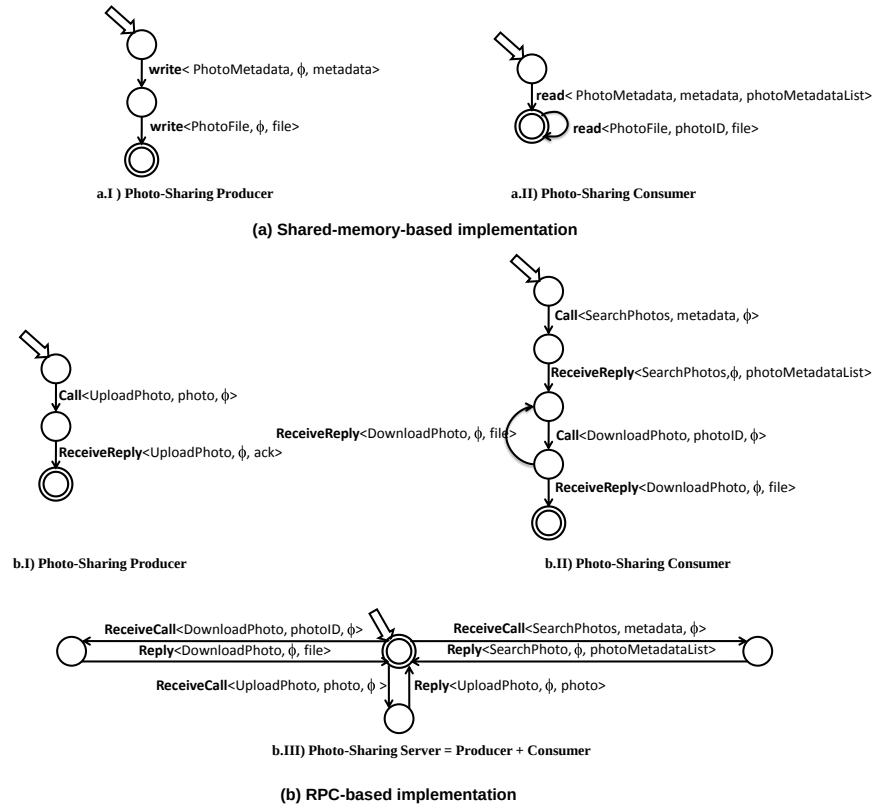


Fig. 5. Heterogeneous protocols in pervasive photo sharing

as *middleware function<application function, input parameters, output parameters>*, where the sets of input/output parameters can be possibly empty (denoted by $\emptyset$). Considering the shared memory implementation, the *Photo-Sharing* producer writes the *metadata* and then the *file* associated with the given photo; while the consumer seeks the list of metadata descriptors (*photoMetadataList*) matching the given *metadata* template using the middleware function *read* and then iteratively reads the files of interest. With respect to the RPC implementation, the producer and consumer exchange photos via the server. The behavior of the producer consists in uploading the photo on the server via the remote call to *UploadPhoto*, and the behavior of the consumer lies in seeking photos of interest (*SearchPhotos*) and then downloading them (*DownloadPhoto*). While the functionalities of the producers and consumers match within the different implementations, there is an obvious behavior (or protocol) mismatch between the RPC-based and shared memory implementations from the application down to the middleware layers. This is such a mismatch that emergent middleware solve, while being synthesized on the fly.

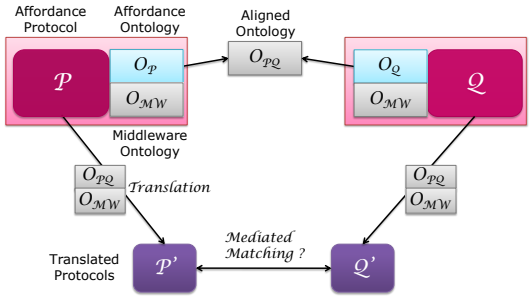


Fig. 6. Approach to emergent middleware synthesis

3.1 Towards On-the-fly Interoperability

Our approach to on-the-fly interoperability, builds on the early theory of application-layer mediators presented in [32]. Figure 6, which we explain after giving its basic ingredients, summarizes the main elements of our approach. We call *affordance*, any functionality that is offered/requested by a networked system, i.e., *Photo-Sharing* in our scenario. In other words, an *affordance* is a high-level action-possibility (or functionality) that characterizes intended and/or possible interactions between the networked system and its environment [14]. Then, given two networked systems that respectively provide and require an affordance that match from a functional standpoint, our goal is to synthesize the mediator that solve the mismatches occurring between the protocols run by the two systems to realize the affordance.

We exploit Labeled Transition Systems (LTSs) to characterize the protocols associated with the realization of affordances. LTSs constitute a widely used model for concurrent computation and are often used as a semantic model for formal behavioral languages such as process algebras. Let Act be the set of observable input/output actions (with output actions being denoted by an overbar) and τ be the silent action. An extended LTS, which makes final states explicit, is a quintuple (S, L, D, F, s_0) where: (i) S is a finite set of states, (ii) $L \subseteq Act \bigcup \{\tau\}$ is a finite set of labels called the alphabet of the LTS, (iii) $D \subseteq S \times L \times S$ is a transition relation, (iv) $F \in S$ is the set of final states, and (v) $s_0 \in S$ is the initial state.

As an illustration, the protocols associated with the *Photo-Sharing* scenario that we informally introduced in Figure 5 actually depict the LTSs of the affordance. In particular, an action of Act is specifically structured as a tuple $(M < A, I, O >)$, where M denotes the middleware function that is called to interact with the peer system through the application function A that is parameterized by input (resp. output) parameters I (resp. O). The set of *middleware*

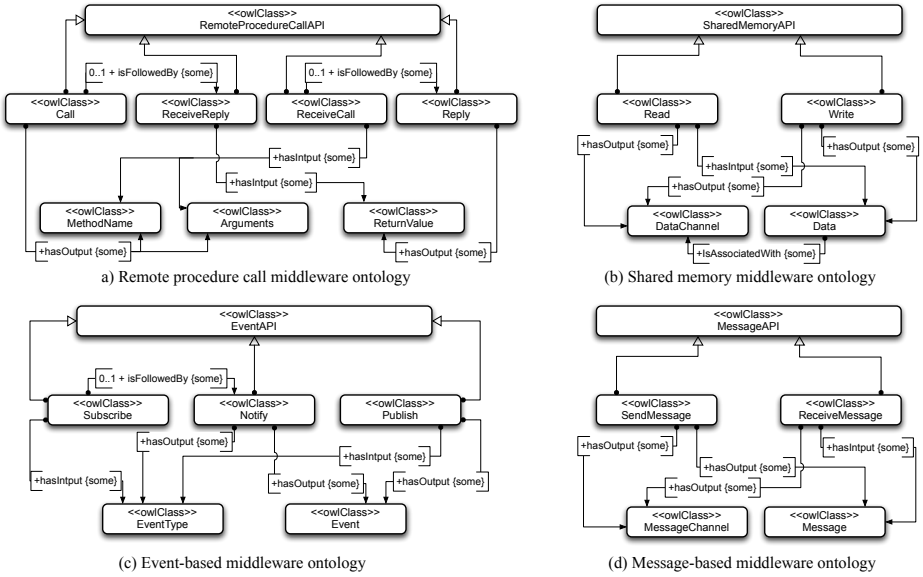


Fig. 7. Reference middleware ontology

functions is defined by the reference middleware ontology $\mathcal{O}_{\mathcal{MW}}$ (§ 3.2) and the one of *application functions* is by the application-specific ontologies $O_{\mathcal{P}}$ and $O_{\mathcal{Q}}$.

Given the two LTSs, $\mathcal{P}$ and $\mathcal{Q}$, characterizing the behaviors of functionally matching affordances, they are translated into LTSs $\mathcal{P}'$ and $\mathcal{Q}'$ for the sake of comparison. The translated protocols are defined in a middleware-agnostic way (§ 3.3) over common application actions from $O_{\mathcal{PQ}}$ following ontology alignment (§ 3.4).

Provided $\mathcal{P}'$ and $\mathcal{Q}'$, we check the *compatibility* of protocols according to the set of traces $T_{\mathcal{P}'}$ and $T_{\mathcal{Q}'}$ associated with $\mathcal{P}'$ and $\mathcal{Q}'$, respectively (§ 3.5). If the two protocols are compatible (also referred to as *mediated matching*), then we are able to synthesize a mediator $\mathcal{M}$ (or emergent middleware) that is such that when building the parallel composition $\mathcal{P}||\mathcal{Q}||\mathcal{M}$, $\mathcal{P}$ and $\mathcal{Q}$ are able to coordinate by reaching their final states.

3.2 Reference Middleware Ontology

State-of-the-art middleware may be categorized according to four *middleware types* regarding provided communication and coordination services [36]: *remote procedure call, shared memory, event-based* and *message-based*. Figure 7 provides an overview of the concepts defining our reference middleware ontology for the four types of middleware, while Section 4 illustrates how they are instantiated with representative middleware solutions. The ontology is given as a set of UML diagrams derived from the specific OWL ontology. In the figure, the ontology

concepts associated with RPC-based middleware includes the *Call* function parameterized by the method name and arguments, which must must be followed by the *ReceiveReply* function to receive the result of the call. Dually, the *ReceiveCall* function to catch an invocation on the server side is followed by the invocation of the *Reply* function to return the result. The ontology of messages for shared memory and message-based middleware is rather straightforward. In the former, the shared memory is accessed through *Read*/*Write* functions parameterized by the associated data and corresponding channel. In the latter, messages are exchanged using the *SendMessage* and *ReceiveMessage* functions parameterized by the actual message and related channel. Regarding event-based middleware, events are published using the *Publish* function parameterized by the specific event while they are consumed through the *Notify* function after registering for the specific event type using the *Subscribe* function.

The proposed reference middleware ontology enriched with specific middleware instances (see § 4 for an illustration) enables dealing with interoperability among middleware instances of the same type, as addressed by the transparent interoperability approach (see § 2.3). However, Interoperability across heterogeneous middleware types requires aligning their respective middleware functions.

3.3 Making Interaction Protocols Middleware Agnostic

We use basic LTS communication actions, i.e., *input* and *output* actions, to align heterogeneous middleware functions of the reference ontology. More specifically, the semantics of the *output* action is the production of an application action in the pervasive network, while that of the *input* action is the consumption of an application action. Then, an *input* and an *output* actions synchronize upon the same application action. The mapping of type-specific middleware actions to the basic LTS communication actions is summarized in Figure 8, thereby enabling to make protocols/LTSs middleware-agnostic. In the figure, the application function is denoted by A and its input (resp. output) parameters are denoted by I (resp. O).

The alignment defined for shared memory and message-based middleware functions is rather straightforward: the *Write* and *SendMessage* functions are defined by the output action while the *Read* and *ReceiveMessage* are defined by the input action. Note that *Read* is possibly parameterized with I if the value to be read shall match some constraints, as, e.g., specified with tuple spaces. The alignment for the event-based middleware functions is straightforward for *Publish*: the publication of the event is defined by the output action. The dual input action is performed by the *Notify* function, which is preceded by at least one invocation of *Subscribe* on the given event[4]. The semantics of RPC functions follows from the fact that it is the server that produces an application action, although this production is called upon by the client. Then, the output action is defined by the execution of *ReceiveCall* followed by *Reply*, while the dual input action is defined by *Call* followed by *ReceivedReply*.

[4] Note that for the sake of conciseness, the figure depicts only the case where a *Subscribe* is followed by a single *Notify*.

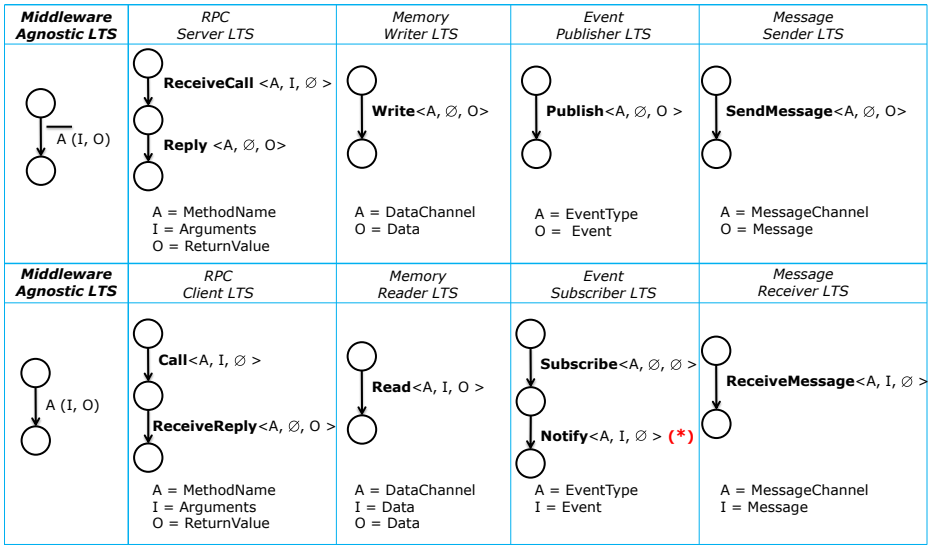


Fig. 8. Semantics of type-specific middleware functions

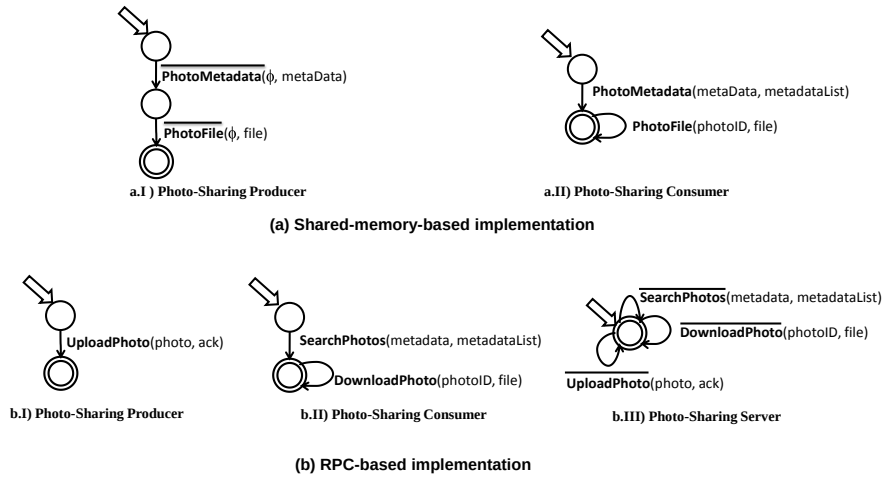


Fig. 9. Middleware-agnostic LTSs of the photo sharing scenario

Given the above semantics of type-specific reference middleware functions in terms of input/output actions, LTSs defining affordance behaviors are made middleware-agnostic. Figure 9 depicts the middleware-agnostic LTSs associated with the photo sharing scenario, which result from the translation of the LTSs of Figure 5 according to the above middleware ontology alignment.

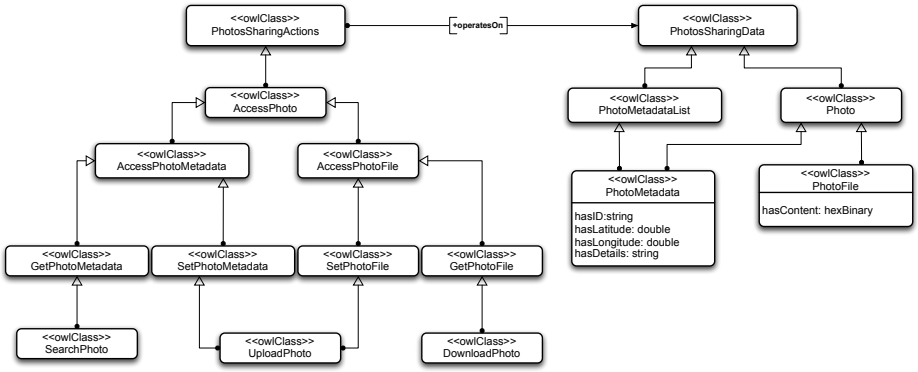


Fig. 10. Aligned application ontology of the *Photo-Sharing* scenario

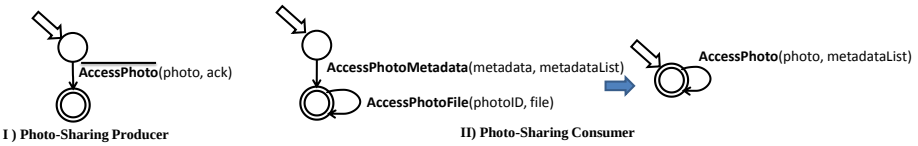


Fig. 11. Middleware-agnostic LTSs with common application actions

3.4 Alignment of Application Actions

In addition to be middleware-agnostic, LTSs must refer to common application actions so as to be able to compare them. Towards this goal, we rely on the alignment of the application-specific ontologies defining the application functions in the LTS. The precise definition of application-specific ontologies that serve our purpose together with that of ontology alignment is part of our future work, where we will exploit research results from the Semantic Web, e.g., see [12].

For illustration, Figure 10 depicts a sample of the aligned application ontology associated with *Photo-Sharing*. The *Photo-Sharing* applications define some *Data* and *Actions* that *operate* on it. The *Data* consist of the *Photo* concept that is defined by some $Metadata$ that specifies the photo's ID, $Location$ based on *Latitude* and *Longitude* and further *Details*, and a $File$ that contains the image. Each data can be accessed through the *Get* and *Set* actions, and other subclass actions (used in the scenario).

Given the aligned application ontology, the middleware-agnostic LTSs are further translated so as to refer to common application actions. More specifically, actions of the LTSs are translated into the closest common application actions (i.e., given by the smallest superclass of the matching actions for each pair of actions). The result is illustrated in Figure 11 for the consumer and producer, leading to identical LTSs for both implementations. Here, notice that specifying a data concept as an action means that full access is provided, which corresponds

to an *Access* action in the application ontology (see the alignment of actions depicted for the consumer).

3.5 Mediated Matching

Given middleware-agnostic LTSs associated with functionally matching affordances, the issue is to synthesize on the fly the mediator (or emergent middleware) that enables the two LTSs to make progress. However, as already mentioned, this is known as a computationally hard problem in general. Then, any solution to automated mediation lies in setting a number of constraints so as to enable devising an efficient algorithm, while solving significant protocol mismatches.

Basically, protocol mismatches decompose into [10]: (i) extra/missing messages, (ii) merging/splitting messages, and (iii) different ordering of messages. The two first mismatches are addressed by the application-specific ontology alignment, which allows relating matching concepts of the application ontology. In our case, this serves identifying relations between application functions and related parameters and thus rewriting LTSs so that they refer to common application actions.

For the third type of mismatch, we impose that any mismatch in the ordering of messages occurs only for non-causally related messages, which we define in terms of the equivalence of traces of the LTSs. In the literature [15], given two processes $\mathcal{P}$ and $\mathcal{Q}$ and two sets $T_{\mathcal{P}}$ and $T_{\mathcal{Q}}$ which denote the set of traces of $\mathcal{P}$ and $\mathcal{Q}$ respectively, $\mathcal{P}$ and $\mathcal{Q}$ are said trace equivalent if $T_{\mathcal{P}} = T_{\mathcal{Q}}$. It has to be noticed that the definition considers $\mathcal{P}$ and $\mathcal{Q}$ defined on the same alphabet and refers to traces that are the very same while we consider to have different alphabets and possibly different ordering of non-causally dependent actions on the traces. Hence, we say that there is a mediated matching between 2 middleware-agnostic LTSs, $\mathcal{P}'$ and $\mathcal{Q}'$, if there exist a trace of $\mathcal{P}'$ and a trace of $\mathcal{Q}'$ that are equivalent modulo the re-ordering of non-causally dependent actions and for any two matching actions, one is an input and the other is an output.

4 From Theory to Practice

While the full deployment of our approach to on-the-fly interoperability is a long-term research challenge, this section discusses practical application of the proposed theoretical approach, considering concrete implementation of the photo sharing scenarios using actual middleware technologies, i.e., SOAP-based RPC and LIME [24] shared memory middleware.

4.1 SOAP and LIME Implementations of Photo-Sharing

We assume that the networked systems running either of SOAP-based and LIME implementations of the *Photo-Sharing* affordance use some discovery protocol to

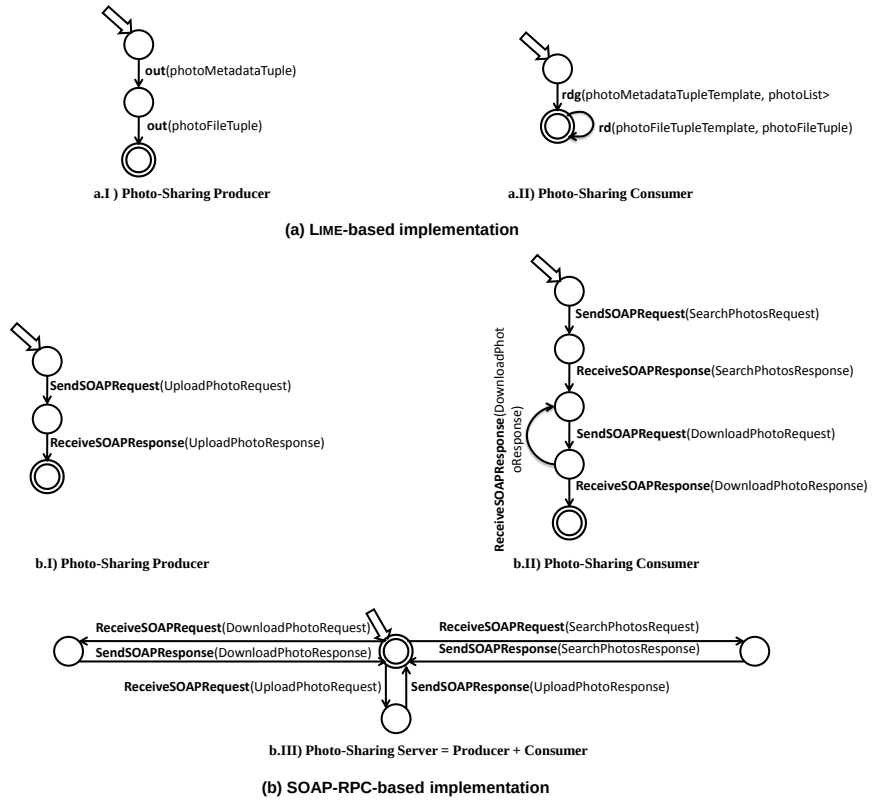
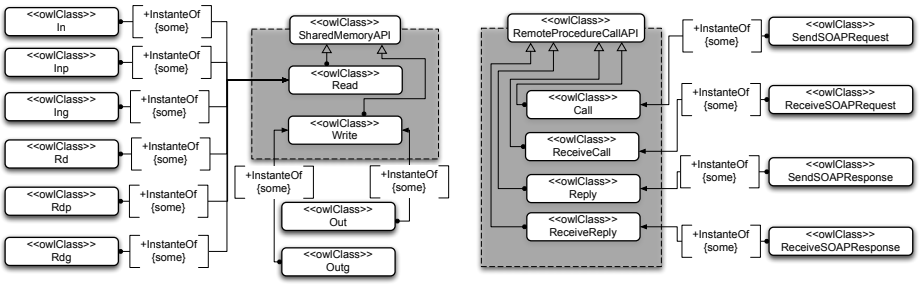


Fig. 12. Photo-Sharing implementations using LIME and SOAP

discover each other on the fly. Interoperable service discovery may then be realized by coupling state-of-the-art solutions to discovery protocol interoperability [7] with interoperable semantic and syntactic service description [2]. We are currently developing such a support where affordance matching from a functional standpoint relies on the description of affordances provided/required by networked systems, using an ontology concept and possible input and output parameters.

Given affordance matching, we need to identify and possibly overcome behavioral (protocol) mismatches. Figure 12 depicts in the form of LTSs, the behavior of the *Photo-Sharing* affordances for the LIME and SOAP implementations, which we assume to be either provided with the interface description at the time of service discovery as promoted by service-oriented computing, or learned on the fly [3]. The actions of the LTSs refer to middleware-specific functions whose semantics in terms of our reference middleware ontology is given in Figure 13.

In the SOAP implementation, data are carried upon *SOAPRequest* and *SOAPResponse* that are generated using the discovered WSDL. In LIME, data



(a) LIME ontology instantiation (b) SOAP ontology instantiation

Fig. 13. LIME and SOAP ontology instantiation

are embedded in *tuples*. A tuple is a sequence of typed fields and contains the information being communicated. Tuples are anonymous, thus their selection takes place through pattern matching on the tuple content. A template or pattern fields contain either *actuals* or *formals*; actuals are values whereas formals act like "wild cards", and are matched against actuals when selecting a tuple from the tuple space [24].

4.2 From Technology-dependent to Interoperable Protocols

Following the presentation of Section 3, actions of the implementation-specific LTSs are translated into actions of the reference middleware ontology according to the corresponding instantiation. We obtain the LTSs of Figure 5. Then, LTSs are made middleware-agnostic according to the semantics definition of Figure 8, leading to Figure 9. Finally, using the application ontology of Figure 10, application actions are aligned according to their least common concept; the result is the LTSs of Figure 11. While the translation process is rather straightforward, the automation of ontology alignment is a major challenge for us, where we need to assess the proposed reference middleware ontology using a variety of middleware instances and also address application-specific ontology alignment.

4.3 Automating Mediated Matching

To automate reasoning about mediated matching, we build upon state-of-the-art software engineering tools, where we are currently assessing different tools regarding in particular performance. As an illustration, we provide here the assessment using the LTSA tool (`http://www.doc.ic.ac.uk/ltsa/`). LTSA (Labeled Transition System Analyzer) is a model-checking tool for concurrent systems: it verifies that the system behavior satisfies particular safety and progress properties. A system in LTSA is modeled as a set of interacting finite state machines. The properties required for the system are also modeled as state machines. LTSA performs compositional reachability analysis to exhaustively search for violations of the desired properties. More precisely, LTSA uses a textual notation (Finite

State Processes, FSP) [21] to describe the component behavior. The tool allows the LTS corresponding to a FSP specification to be visualized graphically. Moreover, LTSA has an extensible architecture, which allows extra features to be added by means of plugins. This feature might prove to be particularly useful for us in order to integrate the OWL semantic within the verification process.

We specifically check the progress property of the whole system that is the parallel composition of the middleware-agnostic *Photo-Sharing* producer and consumer (see Fig. 11), where the corresponding FSP specification is as follows:

```
PHOTO_SHARING_PRODUCER(I='photo,O='ack) =   ACCESSING,
ACCESSING =   (accessPhoto[I][O] -> STOP).
PHOTO_SHARING_CONSUMER(I='photo,O='metadataList) = ACCESSING,
ACCESSING =   (accessPhoto[I][O] -> ACCESSING).
||PHOTO_SHARING = (PHOTO_SHARING_PRODUCER||PHOTO_SHARING_CONSUMER).
progress PHOTO_SHARE ={accessPhoto}
```

One should notice that in LTSA (as in any process algebra), two processes synchronize on the shared variables whereas we consider only two parties (the interaction with third parties are replaced by τ), which synchronize if: (i) their application actions are complementary (i.e., one is with an overbar and the other without), and (ii) the input and output parameters semantically match according to the definition of [28]. For the specific example, the automated verification takes less than 1 ms, which is encouraging result although this needs to be further investigated with more complex scenarios.

Finally, once mediated matching holds, the emergent middleware is realized by actually translating the protocols, which lies in the deployment of mediation engines in the network. This is another of our challenges for which we will build on past experience on transparent interoperability [5].

5 Conclusion

Middleware is in crisis. While it has been successful at sustaining interoperability in distributed systems whose constituent components may be anticipated at design-time, it fails meeting the requirements of today's and future distributed systems that are increasingly complex systems of systems. Complex distributed systems are indeed highly heterogeneous and dynamics, thus challenging any given middleware solution that is domain- and/or technology-specific. We specifically flag the need for *emergent middleware* that is synthesized on the fly as networked systems functionally match and meet. Building upon the rich state-of-the-art interoperability solutions, this paper has discussed the theoretical foundations of the emergent middleware paradigm while significant challenges remain ahead of us, among which most notably turning the theory into a practical framework. The development of the emergent middleware paradigm, from design to prototype implementation is the focus of the European FP7 ICT FET CONNECT project (`http://connect-forever.eu/`), which brings together experts in the fields of middleware and software engineering, semantic modeling of services, protocol learning, and formal foundations of distributed systems.

References

1. Ben Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: EASY: Efficient semantic service discovery in pervasive computing environments with QoS and context support. Journal of Systems and Software 81(5), 785–808 (2008)
2. Ben Mokhtar, S., Raverdy, P.G., Urbieta, A., Speicys Cardoso, R.: Interoperable semantic & syntactic service matching for ambient computing environments. In: Proc. 1st International Workshop on Ad-hoc Ambient Computing (AdhocAmC) (2008), http://hal.archives-ouvertes.fr/inria-00315771/
3. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proc. ESEC/SIGSOFT FSE. pp. 141–150 (2009)
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
5. Bromberg, Y.D., Issarny, V.: INDISS: interoperable discovery system for networked services. In: Proc. ACM/IFIP/USENIX Middleware Conference. pp. 164–183. Springer-Verlag New York, Inc., New York, NY, USA (2005)
6. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Communications 8(1), 127–142 (1990)
7. Caporuscio, M., Raverdy, P.G., Moungla, H., Issarny, V.: ubiSOAP: A service oriented middleware for seamless networking. In: Proc. ICSOC Conference. pp. 195–209 (2008)
8. Cavallaro, L., Nitto, E.D., Pradella, M.: An automatic approach to enable replacement of conversational services. In: Proc. ICSOC/ServiceWave Conference. pp. 159–174 (2009)
9. Chakraborty, D., Joshi, A., Yesha, Y., Finin, T.W.: Toward distributed service discovery in pervasive computing environments. IEEE Trans. Mob. Comput. 5(2), 97–112 (2006)
10. Cimpian, E., Mocan, A.: WSMX process mediation based on choreographies. In: Proc. Business Process Management Workshop. pp. 130–143 (2005)
11. Duftler, M., Mukhi, N., Slominski, A., Slominski, E., Weerawarana, S.: Web Services Invocation Framework (WSIF). In: Proc. OOPSLA Workshop on Object Oriented Web Services (2001)
12. Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag (2007)
13. Flores-Cortés, C.A., Blair, G.S., Grace, P.: An adaptive middleware to overcome service discovery heterogeneity in mobile ad hoc environments. IEEE Distributed Systems Online 8(7) (2007)
14. Gibson, J.J.: The ecological approach to visual perception. Houghton Mifflin (1979)
15. van Glabbeek, R.: The linear time-branching time spectrum i - the semantics of concrete, sequential processes. In: Handbook of Process Algebra, chapter 1. pp. 3–99. Elsevier (2001)
16. Gouda, M.G., Manning, E.G., Yu, Y.T.: On the progress of communications between two finite state machines. Information and Control 63(3), 200–216 (1984)
17. Grace, P., Blair, G., Samuel, S.: A reflective framework for discovery and interaction in heterogeneous mobile environments. ACM SIGMOBILE Mobile Computing and Communications Review (2005)
18. IONA Technologies: OrbixCOMet (1999), http://www.iona.com/support/whitepapers/ocomet-wp.pdf
19. Kiciman, E., Fox, A.: Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In: Proc. 2nd International Symposium on Handheld and Ubiquitous Computing. pp. 211–226 (2000)

20. Limam, N., Ziembicki, J., Ahmed, R., Iraqi, Y., Li, T., Boutaba, R., Cuervo, F.: OSDA: Open service discovery architecture for efficient cross-domain service provisioning. Computer Communications 30(3), 546–563 (2007)
21. Magee, J., Kramer, J.: Concurrency : State models and Java programs. Hoboken (N.J.) : Wiley (2006)
22. Masuoka, R., Parsia, B., Labrou, Y.: Task computing - the semantic Web meets pervasive computing. In: Proc. Semantic Web Conference. pp. 866–881 (2003)
23. McIlraith, S.A., Son, T.C., Zeng, H.: Mobilizing the semantic Web with DAML-enabled Web services. In: SemWeb (2001)
24. Murphy, A.L., Picco, G.P., Roman, G.C.: Lime: A coordination model and middleware supporting mobility of hosts and agents. ACM Trans. Softw. Eng. Methodol. 15(3), 279–328 (2006)
25. Nakazawa, J., Tokuda, H., Edwards, W.K., Ramachandran, U.: A bridging framework for universal interoperability in pervasive systems. In: Proc. ICDCS Conference (2006)
26. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: Proc. WWW Conference. pp. 993–1002 (2007)
27. (OMG): COM/CORBA interworking specification Part A & B (1997)
28. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: International Semantic Web Conference. pp. 333–347 (2002)
29. Ponnekanti, S., Fox, A.: Interoperability among independently evolving Web services. In: Proc. ACM/IFIP/USENIX Middleware Conference. pp. 331–351 (2004)
30. Román, M., Campbell, R.H., Kon, F.: Reflective middleware: From your desk to your hand. IEEE Distributed Systems Online 2(5) (2001)
31. Singh, S., Puradkar, S., Lee, Y.: Ubiquitous computing: Connecting pervasive computing through semantic Web. Inf. Syst. E-Business Management 4(4), 421–439 (2006)
32. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: Proc. WICSA/ECSA Conference. pp. 345–348 (2009)
33. Stollberg, M., Cimpian, E., Mocan, A., Fensel, D.: A semantic web mediation architecture. In: Proc. CSWWS. pp. 3–22 (2006)
34. Sycara, K.P., Vaculín, R.: Process mediation, execution monitoring and recovery for semantic Web services. IEEE Data Eng. Bull. 31(3), 13–17 (2008)
35. Tao, Z., v. Bochmann, G., Dssouli, R.: A formal method for synthesizing optimized protocol converters and its application to mobile data networks. Mob. Netw. Appl. 2(3), 259–269 (1997)
36. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley Publishing (2009)
37. Vaculín, R., Sycara, K.P.: Towards automatic mediation of owl-s process models. In: Proc. ICWS. pp. 1032–1039 (2007)
38. Wiederhold, G., Genesereth, M.R.: The conceptual basis for mediation services. IEEE Expert 12(5), 38–47 (1997)
39. Yellin, D.M., Strom, R.E.: Interfaces, protocols, and the semi-automatic construction of software adaptors. In: Proc. OOPSLA Conference. pp. 176–190 (1994)