Configurable adapters

van den Heuvel, W.J.A.M.; Weigand, H.; Hiel, M.

Published in:
Proceedings of the 9th International Conference on Electronic Commerce (ICEC 2007)

Publication date:
2007

Configurable Adapters: The Substrate of Self-adaptive Web Services

Willem-Jan van den Heuvel
Infolab, Tilburg University
P.O.Box 90153
5000 LE Tilburg, The Netherlands
+31 31 4663020
W.J.A.M.vandenHeuvel@uvt.nl

Hans Weigand
Infolab, Tilburg University
P.O.Box 90153
5000 LE Tilburg, The Netherlands
+31 31 4663020
H.Weigand@uvt.nl

Marcel Hiel
Infolab, Tilburg University
P.O.Box 90153
5000 LE Tilburg, The Netherlands
+31 31 4663020
M.Hiel@uvt.nl

ABSTRACT
With business processes changing constantly, it becomes of crucial importance to equip web services with a series of mechanisms so that they are progressively capable of adapting themselves without any or with very limited human interference. These changes in web service will often lead to interoperability conflicts. To deal with these conflicts, this paper focuses on the use of generic adapters. We show how a generic adapter can solve a number of protocol mismatches, and how adapter configuration can be turned into a feasible task.. For this (re)configuration we look at the field of self-adaptive software.

Categories and Subject Descriptors
D.2.12 [**Software Engineering**]: Interoperability – *Data mapping, distributed objects.*

General Terms
Design

Keywords
Web services, autonomic computing, adapters, process mediation

1. INTRODUCTION
Web-services are rapidly becoming the de facto building blocks of highly distributed business applications operating in widely heterogeneous and possibly short-lived transaction contexts, e.g., in the context of cross-organization business collaborations [16].

ICEC'07, August 19–22, 2007, Minneapolis, Minnesota, USA.

By now, several (competing) specifications, standards and technologies have been developed and marketed in academia and industry, most notably, WSDL to define the interface of web-services, BPEL that allows for orchestration and WS-CDL that enables choreography (conversations between web-services). In fact, these standards are part of a stack of interrelated standards and specifications, which constitute the elementary foundation on top of which individual services ad service applications may be developed and evolved. Actually, many companies are already engaged in SOA project(s), however, mostly at the level of software development projects realizing some local (e.g., departmental or project-related) business processes, but not (yet) at the level large-scale, enterprise-wide business processes [3].

Unfortunately, the enterprise applications that are developed using existing standards and development processes are likely to suffer from several serious shortcomings. Firstly, enterprise applications that are developed by simply placing a simple SOAP layer on top of legacy systems, or selecting and buying a web-service from a (UDDI) repository, and then weaving (hence composing or choreographing) them into a new application are likely to have a fragile architecture and will presumably be hard to adapt and modify to accommodate new or changed business process requirements. Secondly, and related to the previous point, human interaction is needed each time that a change occurs. Since web-services are highly autonomous and loosely coupled web-enabled components, it is very likely that many changes will occur without any prior notification. It is implicitly assumed in many approaches and commercially available tools that human designers/programmers need to be involved each time a change is required.

With the business processes changing constantly, it becomes of crucial importance to equip web-services with a series of mechanisms so that they are progressively capable of adapting themselves, without any or with very limited human interference. Unfortunately, existing techniques for software maintenance cannot be directly used for this purpose because of the following three reasons. Firstly, there exists a lack of control from the perspective of the service consumer over the services that are part of an aggregated web service, as the service realizations remain under control of the service provider. Secondly, as services are highly autonomous and loosely coupled they may evolve independently from each other, resulting in an increased level of complexity and volatility with

which a maintainer has to deal. Thirdly, and lastly, while web-services allow for an isomorphic realization of business services, the evolution of business processes and enterprise applications that are combined with web services can no longer be studied in splendid isolation. Clearly, the evolution of business process and web-services needs to be aligned so that changes at the level of business processes perpetuate to the level of web services, and vise versa.

This greatly increases the level of complexity of (evolving) service-enabled enterprise applications, reinforcing the need for automated support so system programmers can be freed from plumbing and fixing of elementary system-level errors, and focus on "business-level" change management: dynamic reconfiguration on the basis of business concerns.

To ensure that programmers and maintainers can move focus on the business level, issues such interoperability between web services should be solved, so that web services can be easily replaced by others. This idea of service discovery is one of the main ideas behind SOA. However, in searching for a web service the right service might not always be found and if an adequate service is found it could present the output in the wrong format or use a different interaction protocol. To solve these interoperability issues, adapters are used.

In this paper, we work out the concept of the *configurable adapter*. Although adapters have grown to an established technology [18], the idea of configurable adapters is largely unexplored. The research that we base our work on for configurable adapters is that of self-adaptive software, or in our case self-adaptive web services. Self-adaptive web-services are able to reconfigure and re-plan themselves, relying on reflection mechanisms that enable self-evaluation of structure, behavior and goals.

The research questions addressed in this paper are:

- What is a self-adaptive architecture in the context of web services?
- Is it possible to design an adapter that performs process mediation in a generic way?

This paper is structured as follows. In section 2, we will outline self-adaptive systems in a technology-independent way. Section 3 translates this framework to self-adaptive services. A configurable adapter architecture for process mediation is proposed in section 4. We conclude in section 5 with a short discussion of related work, and directions for future research.

2. SELF-ADAPTIVE SYSTEMS

Self-adaptive systems evaluate their behavior using reflection mechanisms, and modify themselves in case that their behavior does not conform to predefined goals, or, when the system can be optimized in terms of non-functional properties such as performance, security and stability. We call that part of the system that is modifiable the adaptable system. Following standard control theory [14], the adapting part is called the controller that changes the adaptable system by means of effectors based on information it collects from sensors. We assume that the information can come from the adaptable system itself (monitoring) or from the environment, which may include peer systems.

In the autonomic computing architecture [6;9] the decision process carried out by the controller is represented by a control loop, which consists of four functions: monitor, analyze, plan and execute. In our conceptualization (Fig. 2), the monitoring is based on *performance indicators*, such as response time or an error raised. Analysis implies that the performance indicator is compared to a given *norm* or threshold. The norms are related to the *goals* of the system, which we assume to be fixed from the perspective of the controller. The relationship between goal and norm may be indirect. For example, the goal may be to adhere to norms specified in service level agreements (SLA's) with peer systems. Then the SLA's, together with their performance norms, may evolve over time, while the goal of the system remains the same.

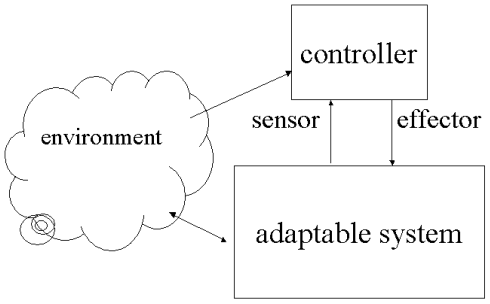

Fig. 1. General architecture of a self-adaptive system

For reasons explained partly below, it is essential that the controller is knowledge-based (cf. [7,4]). There are several ways to realize that, but in our conceptualization, it means that the controller maintains a list of scripts, where a script indicates what to do (change plan) given a certain state, where a state is measured using a *performance indicator* against some *norm*. The task of the planner is to find the best matching script. Finally, the change plan in the script is executed, which may be done partly by means of the effectors on the adaptable system, but also by influencing the environment directly, such as a conversation with peers.

The performance of the system is monitored under a *perpetual testing* strategy [13]. In case the configuration of the adaptable system is not a one-shot event but rather a continuous effort to find the optimal configuration for realizing the goals, we talk about (first-order) self-adaptation. Determination of the configuration can be a random explorative process in the beginning, while gradually becoming more stable. To support such an explorative process – that is, the behavior of the controller is not fixed, but is able to learn – we extend the control loop as defined above with a second loop [19]. This second loop configures the controller on the basis of the fixed goals: it monitors how well the controller is doing in realizing the goals, and has the possibility to modify the controller configuration: since the configuration of the controller is exhaustively represented in the scripts, the task of the learning process is to find the most successful scripts. This learning

process may be supported by imitation, which basically means exchange of successful scripts between peers [8].

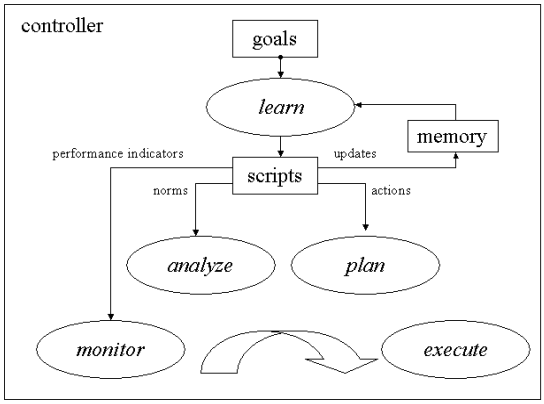

Fig. 2. Architecture of the controller

First-order self-adaptation, even when it includes a learning loop, is bound to limitations, as the goals may become infeasible. Hence it may become necessary to adapt the goals. This requires a *second-order* adaptive system (double-loop learning, [1]). Although this kind of adaptation is often projected inside the adaptive system, thereby raising its complexity, we model it as a combination of two systems, where the higher system (manager) monitors the behavior of the lower system (agent) and subsequently may adapt the goals of the agent. The manager also has its own goals that it cannot change, and is itself an agent of a higher-level system, ultimately the human user.

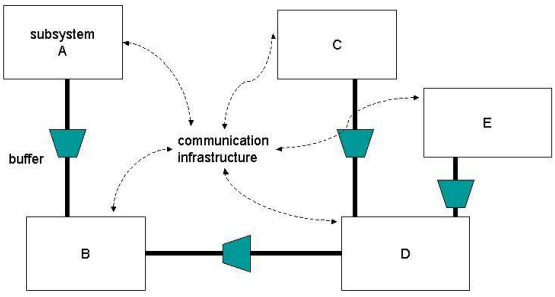

Fig. 3 Architecture of self-adaptive society

The self-adaptive system perspective described so far is very general and provides a good basis for developing self-adaptive web services. However, web services, like agents and many other information systems, are distributed in nature. Therefore the system perspective should be supplemented with a global system, or society view. For a society (Fig. 3) consisting of multiple adaptive subsystems there are a few essential basic rules:

(1) adaptation of the society is done typically by *rearrangement* of the subsystems, or the replacement of one subsystem by another one. In other words, the most important object of adaptation is made up by the subsystem interconnections;

(2) for the society it is not only important to accommodate necessary adaptations, but also to avoid unnecessary adaptations. Since subsystems may change autonomously, it is important to have some kind of *buffers* between subsystems to reduce the effects of a certain change as much as possible. The function of buffers is filled in by *adapters* in service-oriented computing.

(3) For the society it is important to support the *dissemination* of successful adaptations. Subsystems can imitate one another. The society should incorporate mechanisms to support this, such as a shared language and shared repositories.

3. SELF-ADAPTIVE WEB SERVICES

Translating the general self-adaptation model to the area of web services [16] leads to a distinction between a *managed service* and a *service manager*. The managed service can be atomic or composite and is controlled by the service manager through standard management interfaces called touch points [9], corresponding to the sensors and effectors above. Several standards have been suggested to standardize the communication between managed web services and controllers, through management interfaces [15]. Notably, the Web Services Distributed Management (WSDM) specification aims at interoperability between managed services (defined using WS-Resource and WS-ResourceSpecification) and service manager focusing at two distinct, but related, aspects: Management of Web Services (MOWS) and Management Using Web Services (MUWS) [12]. While web-service management standards such as WSDM do offer a vendor-neutral standard for management of service resources, they are not specifically designed for supporting *self-adaptive* web-services.

3.1 Interoperability Conflicts

The self-adaptive web service is able to deal with changes. Here we specifically study changes that have impact on the interoperability of the services. As services are usually integrated to form an application is of importance that these services remain compliant. Severe interoperability conflicts may arise in case the interface of a web-service in the provided or required environment is (unilaterally) altered. We have distinguished three levels of conflicts, namely signature, protocol and quality-related conflicts.

3.1.1.1 Signature-Level Conflicts

The following interoperability problems between provided and required signatures may occur:

1. Input message mismatches
2. Output message mismatches
3. Operation mismatches
4. Network protocol mismatches

The first three types of interoperability conflicts pertain to the logical part of a WSDL definition, the latter type to the physical definition. These interoperability problems may occur regardless of the granularity of a web-services: both in case of an atomic and composite web-service these conflicts may arise, at least, if only the provided context in which a web-service operates is taken into consideration.

3.1.2 Protocol-Level Conflicts

Protocol-level interoperability conflicts occur in case at least one invocation sequence of a required protocol of a web-service, `A`, is not accommodated by the provided interface of another (supplier) web-service, `B`, or vise versa [17].

This category of interoperability conflicts may happen in various situations:

- Missing signatures: this situation occurs if the web-service A tries to invoke a signature (port type), e.g., in the context of a business process, which is not realized by web-service B.
- Superfluous port type invocations: the web-service A in fact invokes port types of web-service B in the right sequence, however, web-service B expected web-service A to invoke more of its port types.
- Sequencing interoperability conflicts: at least in one specific situation, the order in which web-service A invokes the port types of web-service B is not correct.
- Synchronization interoperability conflicts: even in case the required protocol of web-service, A, and the provided protocol are equivalent, interoperability conflicts may occur due to differences in timing (joining and forking).

3.1.3 Quality-related Conflicts

Quality-related conflicts may arise in case that specific quality of service related characteristics, which were advertised in the interface of a web-service, A, are not realized, while these characteristics were actually expected and required by a consuming web-service, B.

In the context of component-based development, quality conflicts have been studied at runtime or design-time (inspection time). Availability, performance, reliability and usability are well-known examples of runtime quality attributes, while design-time quality is often captured concentrating on issues like, conceptual integrity, portability, security and testability [10]. For the purpose of web-services, we may extend and refine this categorization to cater for key requirements stemming from the SOA, including, payment models, billing models, transaction models, and coordination models.

Taking into consideration the above quality-related aspects, we have identified the following two quality related conflicts:

- *Non-compatible quality dimensions*: this may happen in case a web-service expects a quality attribute to be specified in a certain measurement, while a supplier web-service has actually declared a quality attribute using a non-compatible measurement.
- *Dissimilar quality values*: when assuming that A and B adopt equivalent quality dimensions, incompatibilities can arise if B's service implementation simply does not deliver the quality A necessitates.

Apart from interoperability conflicts, there is of course the inherent tension between interests of the service provider and the service client. The tension is usually resolved in an SLA or WS-Policy agreement. The service provider should be able to adjust the service level when the client requires so, and to maintain an agreed upon service level by means of continuous monitoring.

3.2 Adaptation strategies

If faced with a conflict then the web service needs to respond to this. We distinguish two strategies of self-adaptation that could be used in web services:

Internal adaptation. By definition, a web service is designed for multiple usage contexts, and will consequently support internal adaptation to some extent. When only the interface of the managed service is changed, we talk about black box adaptation. Black-box adaptation does not demand complete insight into source-code of web-services. That does not mean that a black-box adaptation mechanism does not touch or change the code of a web-service; on the contrary, some black-box adaptation techniques require some service code to be modified, e.g., to add some gluing scripts. Changes to the internal workings of the managed service are called white box adaptations, necessitating a deep understanding of the code, its architecture and working.

Mediation. This strategy involves a mediator or adapter between the service and the related service and is particularly effective in case the service requester and provider are not equipped with dynamic communicational capabilities, or service policies and contracts cannot be renegotiated, e.g., since this would incur high transaction costs. In this strategy, interoperability conflicts are resolved using a service adapter, which allows mapping the interfaces (signatures, protocols and QoS interfaces) of a service requester to that of a service provider. Introducing an adapter typically has a performance penalty, but an advantage is that the autonomy of the services increases. If the adapter is under the direct control of the service manager, the mediation strategy could be regarded as a special case of internal adaptation, but as this is not necessarily the case, we treat it separately.

Mediation as an adaptation strategy can be needed when setting up a new collaboration with some service, or at run-time, when a service in an existing configuration unilaterally changes its interface. In the first case, the following tasks need to be performed:

1. Discover a service that offers the functionality required and whose provided interface matches above some threshold value
2. Assess the nature of the mismatch between the required service and the provided service
3. Select a generic adapter from the service repository
4. Configure the generic adapter(s)
5. Deploy

In the second case, the first step can be omitted; the rest of the process is the same.

4. GENERIC PROTOCOL ADAPTER

Services in the SOA framework are designed to be open and possibly without knowledge at design time about the type and number of clients that will access them. The possible interactions that a web service can support are likewise specified at design time, using what is called a business protocol or conversation protocol [2]. In the following, we assume that not only the service in question specifies a provided protocol, but that also the clients specify a required protocol. The role of the adapter is to align the two, so that neither the client nor the service needs to be adapted.

A protocol consists two parts: M, a set of message definitions (in-going and out-going), and C, a context specification. For the message definitions, we simply use the input and output definitions from the WSDL interface. The context specification contains several parts:

- *meta-information* about the protocol such as a reference to a standard on which the protocol is based;
- *ordering constraints* on the messages (m1 < m2) and
- *context variables* whose values can be used as defaults. The meta-information may embed the context in another published context, implying that the variable settings are inherited (default inheritance) to the present context.

The *information space* of a protocol is the set of information units, where the information units are the message fields (from the messages in M) prefixed with the name of the message and the role (P for provided, R for required). For example, if the message "quote_request" contains a field

```
customer.address.zipcode
```

then

```
R.quote_request.customer.address.zipcode
```

may be an element of the information space.

A protocol mapping defines a translation from one protocol to another. Central to the protocol mapping is a mapping between the provided and required information spaces. We propose to define this mapping as a generalization of an XSLT/XQuery mapping between messages, as this would allow the incorporate the full current power of these techniques. As far as the mapping between information units is concerned, we may use an ontology-based approach, such as the WSMX Runtime Data Mediator described in [5], but for the process mediation described below any other mapping approach could be used instead.

If the two protocols are the same, then the mapping is the Identity function. The mapping specification consists of XQuery statements between information units. If no mapping specification is given for a certain information unit, the Identity function is assumed. If an information unit in the required message does not have a corresponding information unit in the provided message, and no mapping specification is given, the default value given in the context of the sender is taken as a constant. If there is none, the default value given in the context of the receiver is taken.

Example. Protocol P assumes an XML-based order_request defined concisely as on the left below, whereas Protocol R assumes a flat message given on the right.

```
P.Order_request
       R.Order_request
[Customer
       [Name
       Name
       Address
       Id
       Zipcode
Address
       Product_name
       Street
       Product_quantity]
       Number
       Zipcode
       Citycode
       Country
Product
       Id
       Number
       Order_date]
```

The following mapping rules are supposed to map the data from the information items required by client R to the ones provided by service P. To keep the example concise, we omit the prefixes.

```
Customer.Name ← Name
Customer.Id ←
       XQuery_Transform(Name,Registry)
Address.street ← XQuery_Transform(Address)
Address.number ← XQuery_Transform(Address)
Address.zipcode ← Zipcode
Address.citycode ←
       XQuery_Transform(Zipcode, Registry)
Address.country  ← $COUNTRY
Product.id ← Product_name
Product.number ← Product_quantity
Order.date ← current_date()
```

The intended meaning is supposed to be straightforward in most cases, but some lines need more attention. The `order_date` is not provided by P2, but by standard function `current_date()`. The address field must be split up in a street field and a number field; this is done by means of two transformations (basically, tail and head). Another transformation is used to convert a zipcode into a city code. The `customer_id` is not found in the order_request of R, but we assume that customers register first and then get a unique

customer id, and the mapping calls upon the register service to look up the customer and get the id.

4.1 Flexible Protocol Adapter

The Flexible Protocol Adapter FPA implements an adapter between two services on the basis of a requested and provided protocol and a mapping specification. The FPA is a combination of a mapper and a spooler. It contains basically two processes: fetch and forward, and utilizes a mapper function that works on the basis of mapping rules per information item. The following pseudo-code should convey the essence.

```
fetch = while () {
       m= get.inputmessage();
       S = information_units(m);
       store (S,Buffer);
       }

forward = while () {
  forall P in (Service,Client) {
       E = expected_messages(P);
       forall m in E
          if check_input(m)
              {x = create_instance (m);
               mapper(x);
               put_outputmessage(x, P);
              }
       }
       }

mapper(message m) =  {
  forall ToInfItem.name in m {
      r = get_rule(ToInfItem.name);
      m.ToInfItem.value = apply(r,Buffer)};
       }
```

Next to the mapping and spooling the FPA is able to solve the protocol level conflicts discussed in Section 3.1. In [2] and [5], these conflicts are called mismatches and consist of:

- Message order mismatch
- Extra message mismatch
- Missing message mismatch
- Message split mismatch
- Message merge mismatch

In some cases, the mismatches are irresolvable because of ordering constraints. However, if there is a solution, the mismatch is solved by the FPA, as can be validated. For the mismatches at the operation level, such as signature mismatch or constraint mismatch, FPA uses essentially the same approach as [2], that is, the specification of XQuery transformations.

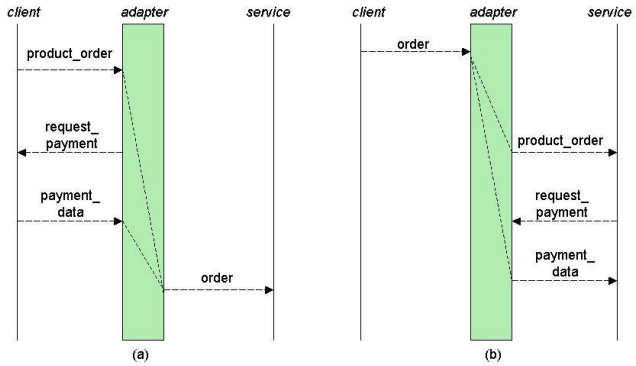

Fig. 4. Message split and merge mismatch

Example. Assume a product order protocol and suppose that the provided interface specifies one message "order" that includes a product id and the creditcard data of the customer, whereas the required interface includes two messages, "product_order" and "payment_data". The "payment data" message is sent after a "request_payment" is received. When the client sends the product order message, the adapter extracts the information items from it, but it cannot create an order message, as the creditcard data are missing. However, it can create a "request_payment" message for the client that triggers the client to send the payment data. Now the adapter can create the order message and send it to the service (Fig 4a). The reverse case is that the required interface contains one message and the provided interface two. Upon receiving the single order message, the adapter creates a product order message first, sends it to the service, and then creates and sends the payment message (Fig 4b).

4.2 Adapter configuration

The FPA is a flexible generic adapter, but its effectiveness depends critically on the availability of a mapping between the two protocols that need to be aligned. To prevent that the adapter will become the next bottleneck in adaptability of a integrated network or service, the adapter itself should also be adaptable. We can therefore make the separation between the service manager (controller) and the managed service (adaptable system). The managed service contains the mappings between protocols and the service manager task is to maintain (adapt) these mappings.

The service manager itself cannot construe this mapping. However, what it could do is *configure* a mapping on the basis of one or more available mappings (Fig.5). Assume that both the required and the provided protocol are both set up as instantiations of standard protocols, e.g. Rosettanet, SAP, or HL7 (health domain). And assume that in both cases, the instantiation itself is documented in the form of a mapping between the protocol as used and the standard from which it is derived. In that case, it is possible for the service manager to search for a mapping between the two standards involved. It is reasonable to expect that mappings between the most common protocols will be provided by standardization organizations or on a commercial basis. Then the service manager can configure an adapter as a composition of mappings.

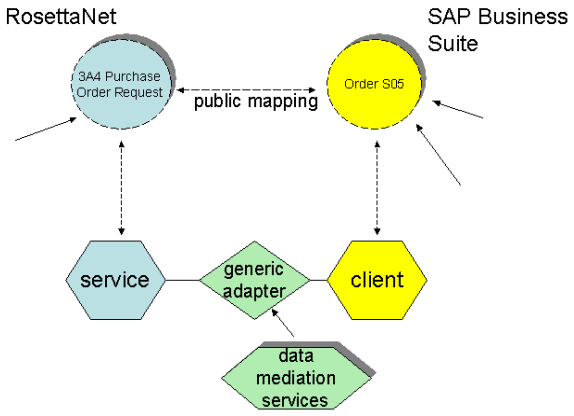

Fig. 5. Adapter configuration

In general, a mapping between two protocols is not a monolithic, but draws itself on smaller specialized adapters, for example, a currency converter, or a bank account/IBAN converter. Typically, these smaller adapters do not provide process mediation but data mediation only. Supposedly, these adapters will be made available as services on the web, and as such they can be called by the generic adapter as generated by the service manager. The adapter that provides the mapping between two protocols would then itself become a composite web service, which can be subject to internal adaptation (or reconfiguration).

As long as adapters and mappings are well-defined in terms of their input and output, referring directly or indirectly to standard protocols or ontologies, the adapter configuration is a non-trivial but nevertheless feasible task.

5. CONCLUSION

In this paper, we have explored self-adaptation of web services, a topic that is growing in relevance rapidly. We have shown how a general self-adaptation model can be realized in the Service Oriented Architecture. One important adaptation strategy is mediation by means of configurable generic adapters. We have described the adapter adoption process and how a generic adapter can be configured for process mediation.

There are several tools that ease the development of XQuery mappings such as Contivo (www.contivo.com). However, these tools typically address the message level rather than the protocol level and assume a human designer rather than self-adaptation.

Our research builds on previous work by [2]. This work describes possible mismatches between web services and solution patterns that can be used to resolve the mismatches. Our solution addresses the same kind of mismatches, but rather than providing patterns that can be applied by a human designer, we have described a generic adapter that incorporates these patterns automatically.

A similar list of mismatches has been described independently in the context of the WSMX/WSMO working group. [5] describe a Process Mediator that acts on public processes (represented as WSMO choreographies) of the parties involved in a communication and adjust the bi-directional flow of messages to suit the requested/expected behavior of each party, very similar to the FPA. The approach assumes that choreographies are described in the WSMO ontology. Although an ontological level as added-value, we do not want to require such a framework given the present state-of-the-art, and prefer to build forth on industry standards as XQuery. Other differences between our FPA and the Process Mediator are that we represent a protocol by means of messages and ordering constraints, rather than a process description, and that we also utilize a context to fill in missing values.

An important issue for future research is to evaluate the solution direction given in this paper by building a prototype generic adapter and by testing the adapter configuration on realistic scenarios.

6. REFERENCES

[1] Argyris, C. and Schön, D. *Theory in practice: Increasing professional effectiveness*, San Francisco: Jossey-Bass, 1974.

[2] Benatallah, B. F. Casati, D. Grigori, H. R. Motahari Nezhad, F. Toumani, Developing Adapters for Web Services Integration., Proc. CAiSE, 2005, pp.415-429.

[3] Spott, D., The Business Case for Service Oriented Architecture, *CBDI Journal*, pp. 3-11, November 2004.

[4] Cheng, S-W., Huang, A-C., Garlan, D., Schmerl, B., and Steenkiste, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37, 10, October 2004.

[5] Cimpian, E., Mocan, A. WSMX Process Mediation Based on Choreographies. Proc. 1st Int Workshop on Web-Service Choreography and Orchestration for Business Process Management (BPM 2005), Nancy, 2005.

[6] Ganek, A.G., T.A. Corbi, The Dawning of the Autonomic Computer Area, *IBM Systems Journal*, 42 (1): 5-18, 2003.

[7] Georgas, J., R. Taylor, Towards a knowledge-based approach to architectural adaptation management. *Proc. WSOSS'04*, ACM Press, 2004, pp.59-63.

[8] Hiel, M., H. Weigand, Requirements on the Use of Goal-Directed Imitation for Self-Adaptation. In: *17th Int. Workshop on Database and Expert Systems Applications (DEXA 2006)*, IEEE Computer Society 2006, pp. 98-102.

[9] Kephart, O., D.M. Chess ,The Vision of Autonomic Computing. *Computer* 36(1):41-50, 2003.

[10] McGovern, J., Sims, O., Jain, A., Little, M., *Enterprise Service Oriented Architectures Concepts, Challenges, Recommendations.* Springer, 2006.

[11] Mocan, A, Cimpian, E. WSMX Data Mediation. Technical Report WSMX Working Draft, http://www.wsmo.org/TR/d14/v0.1/ , March 2005.

[12] OASIS, 2006. "An Introduction to WSDM", Committee Draft, Report nr.: WSDM-1.0-intro-primer-cd-01, 24-2-2006.

[13] Osterweil, L.J., L.A. Clarke, D.J. Richardson, and M. Young, Perpetual Testing, In: *Proceedings of the Ninth International Software Quality Week*, 1996.

[14] Owens, D.H. *Feedback and Multivariate Systems*. Peter Peregrinus Ltd, Stevenage, 1978.

[15] Papazoglou, M.P. and W.J. van den Heuvel, Web Services Management: A Survey, *IEEE Internet Computing*, 9(6):58-64, 2005.

[16] Papazoglou, M.P. and D. Georgakopoulos, Service-Oriented Computing. In: *Communications of the ACM*, 46(10):25-28, October 2003.

[17] Reussner, R., Parameterised Contracts for Adapting Software Component Protocols, PhD Thesis, Karlsruhe University, 2002.

[18] Yellin, D. M., Strom, R. E.: Protocol specification and Component adaptors. *ACM TOPLAS*, vol. 19, no. 2 (1997).

[19] Weigand, H., W.J. van den Heuvel, The challenge of self-adaptive systems for E-commerce. In: *Group Decision and Negotiation (2007).*