

Structural models of patterns of message interchange in decoupled hypermedia systems

Sinuhe Arroyo
Digital Enterprise Research Institute
Technikerstrasse, 21a
A-6020 Innsbruck, Austria
+43 512 394545
sinuhe.arroyo@uibk.ac.at

Miguel-Angel Sicilia
University of Alcalá.
Plaza de San Diego s/n
28801, Alcalá de Henares, Madrid
+34 91 8866663
msicilia@uah.es

José-Manuel López-Cobos
Atos Origin SAE
Albasanz, 12
28037 Madrid, Spain
+34912148612
jose.lopez@atosorigin.com

ABSTRACT

Open hypermedia systems provide a decoupled approach to structural computing. This results in architectures in which user agents as browsers or other intermediate entities build the structure of hypermedia by communicating with distributed servers. This in turn raises the consideration of the different message exchange patterns that may be implemented, and the resulting behavior of systems according to them. This paper describes an structural model for such kind of message patterns, and provides a list of possible courses of interaction. Then, the influence of such heterogeneity of communication patterns is analyzed from the perspective of composing structure in hypermedia. This represents a first attempt to advance current Service-Oriented specifications for decoupled hypermedia to cover the heterogeneity of potential system behaviors.

Categories and Subject Descriptors

H.5.4 [Information Interfaces and Presentation]:
Hypertext/Hypermedia – architectures, theory, user issues

D.m [Software]: Miscellaneous

General Terms

Design, Standardization.

Keywords

Service-oriented computing, choreography.

1. INTRODUCTION

Structural computing is an approach to service provision that grew out of the interoperability work of the *Open Hypermedia Systems* community [10]. Further research has addressed the implementation of such services on common *Web Service* infrastructure [9]. The system-to-system interaction assumed in such frameworks is resolved in terms of single request-responses to services. Nonetheless, the assumption that link servers and other services will be available continuously for synchronous communication is far from being realistic. On the contrary, the structural philosophy is that of adding structure to hypermedia nodes as rendered on user agents, so that there's nothing that prevents that such structure could be rendered asynchronously, at the moments in which some servers respond. In addition, it may be possible that the reception of structure information results in a

new request. For example, the Auld Leaky¹ server implements *behaviours* in the following form "behaviour remains opaque to Auld Leaky and it is left to the client to implement the events of the behaviour object". Then, if we have a concrete behaviour attached to a data object as the following:

```
<behaviour event="display">  
  <behaviourvalue key="context.detail">high  
</behaviourvalue>  
<behaviourvalue key="context.about">Pio Moa  
</behaviourvalue>  
</behaviour>
```

The "context.detail" value may be interpreted by the client as an indication to initiate a new query for links in the available linkserver(s) to add more structure to the current node, considering the historian "Pio Moa" as the point of reference of links retrieved. Even though this kind of chained interactions is out of the scope of the current reference models as FOHM [11], this type of architectural issues have an impact in the final behavior of user agents, and these should be included in future open hypermedia models for coherence in user agent behaviour. This paper describes an structural model for such patterns (described as choreographies) and its potential applications to the architecture of structural computing solutions.

A *choreography* describes the external visible behavior of services as message exchanges. A considerable number of approaches exist such as BPEL [5], WS-CDL [7] or WSCI [2] or WSMO – Choreography [9], which can be used to model the external visible behavior of services. However, they constitute incomplete solutions, either because they do not include proper support for semantics (BPEL4WS, WSCI, WS-CDL²), they have a lack of technological independence (BPEL4WS, WS-CDL), they mix internal and external aspects (BPEL4WS, WS-CDL) and finally, they do not provide consistent approaches or present *ad-hoc* ones to solve behavioral and structural heterogeneities (BPEL4WS, WS-CDL, WSCI or WSMO-Choreography), or in some cases, they mix both aspects resulting in confusing specifications (BPEL4WS, WS-CDL, WSCI or WSMO-Choreography).

When designing a conceptual framework for choreography, a number of models need to be taken under consideration. In a first cut it is easy to differentiate among the semantic and syntactic aspects. When going into details, the syntactical model is further

¹ <http://www.equator.ecs.soton.ac.uk/technology/linky/>

² It supports the recording of semantics, but it does not really use them at all.

subdivided to cover structural, behavioral and operational aspects. While structural aspects allow defining abstraction that allow describing a problem domain [1], behavioral aspects depict the way parties interact with the others, and finally operational ones depict the way interaction among heterogeneous behavioral styles can take place. This paper describes the structural model of SOPHIE³, a conceptual framework that attempts to overcome the limitations of existing technology, and its application to decoupled hypermedia applications. SOPHIE takes as core element for describing the interaction among parties MEPs. As messages are passed, a variety of scenarios can be played out. MEPs allow to model the sequence and cardinality of messages, defining the order on which parties send a receive messages. As parties follow different Message Exchange Patterns (MEPs) in their interactions, the problem of solving the mismatches in regard the names and types of the elements exchanged and the number and sequence of messages, becomes more affordable and reduced to map the MEP of interacting parties. In doing so, MEP need to be describe following some sort of formalism such as ASMs [4] or Petri nets. If we take a step further, and describe this patterns following the ideas posed by the Semantic Web [3], the problem is reduce to the ontological mapping of the different interaction procedures. The application of MEPs to existing open hypermedia models results in richer user agent-server interaction patterns that should be integrated in reference models as a way to advance standardization.

SOPHIE elaborates on the work conducted on [6], extending it by the addition of patterns that cover some binding-specific information. In this sense, it adds patterns that take under consideration faults, acknowledgements, timeouts and correlation among messages. It is important to notice that this work does not make any assumptions on the underlying communication frameworks (i.e. SOAP, WSDL), as in designing a conceptual model this type of considerations must be left aside.

The rest of this paper is structured as follows. Section 2 provides the basics of the structural model. Then, a catalog of common MEPS is described in Section 3. In Section 4, a sketch of possible user behaviors related to the MEPs is provided. Finally, conclusions and future research directions are provided in Section 5.

2. STRUCTURAL MODEL

The structural model deals with the provision of a reusable collection of entities following different levels of abstraction that facilitate the basis for the description of a conceptual model. Table 1, enumerates the entities that allow the structural model to be defined.

Conversations are the outer most entity of the structural model. They represent the logical entity that permits to group a set of related message exchanges among parties. Conversations are composed of a set of building blocks. *Elements* describe elementary units of data that define a name, a type⁴ and a value

³ SOPHIE is an acronym for Semantic web services chOreograPHI engInE

⁴ Element types are defined by the standard XSD [8]

that build documents. Documents are complete, self-contained groups of elements (in open hypermedia systems, these may be contextual queries as those of Auld Leaky).

Table 1. Structural model entities

$party = [name, URI, role]$
$element = [name, type, value]$
$document = [name, URI, elements^{*5}]$
$message = [name, URI, from^{?6}, to^{?}, documents^{*}]$
$messageExchangePattern = [name, URI, description^{?}]$
$message\ exchange = [name, URI, mep^{?}, messages^{*}]$
$conversation = [name, URI, messageExchanges^{*}]$

Documents are transmitted over the wire within *messages*. Messages characterize pieces of information that can be exchanged among parties. As messages are exchanged, a variety of recurrent scenarios can be played out as defined by *Message Exchange Patterns (MEP)*. A MEP defines a minimal contract among parties. They allow the sequence and cardinality of messages to be modeled, defining the in order which parties send and receive messages. The constituent *description* is an Abstract State Machine (ASM) that depicts the behavior of the pattern. A set of messages sent and received among parties optionally following a Message Exchanged Pattern that account for a well defined part of a conversation, is referred to as a *message exchange*. A *conversation* can thus be defined as a set of message exchanges optionally following message exchange patterns, among *parties*⁷. Every conversation need to rely on top of some communication facility, referred to as a *communication network*.

3. DESCRIPTION OF MEPS

A number of patterns are defined in [6] which are further extended in this work with the addition of patterns that try to cover a broader spectrum of message exchanges. They take under consideration some binding-specific information, such as faults, acknowledgements, timeouts and correlation among messages. Nevertheless this extension always keeps in mind that message exchange patterns identify a minimal contract between parties, and should contain only information that is relevant to the interacting parties.

2.1 In-Resend-On-Fault

This pattern consists of exactly three messages as follows:

1. A message:

⁵ The symbol "*" represents that there can exist zero or more instances of the attribute

⁶ The symbol "?" represents zero or one instances of the attribute

⁷ Where, *role* specifies whether the party is an *initiating party* or *answering services*. Generally speaking the role of the parties is subject to be changed as the conversation evolves. However, for consistency, during a particular message exchange, the roles of the interacting parties should be kept.

- indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. A fault message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
 3. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-resend-on-fault/".

2.2 In-Optional-Resend-On-Fault

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. A fault message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
3. An optional message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-optional-resend-on-fault/".

2.3 Out-Resend-On-Fault

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. A fault message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
3. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-resend-on-fault/".

2.4 Out-Optional-Resend-On-Fault

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. A fault message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
3. An optional message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-optional-resend-on-fault/".

2.5 In-Multi-Out

This pattern consists of an undetermined number of messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. An undetermined number of messages:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-multi-out/".

2.6 Out-Multi-In

This pattern consists of an undetermined number of messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. An undetermined number of messages:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-multi-in/".

2.7 In-On-Ack

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

2. An ack message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
3. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-on-ack/".

2.8 In-Optional-On-Ack

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. An ack message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
3. An optional message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-optional-on-ack/".

2.9 Out-On-Ack

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. An ack message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
3. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-on-ack/".

2.10 Out-Optional-On-Ack

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

- indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
- sent to node N

2. An ack message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
3. An optional message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-optional-on-ack/".

2.11 In-On-Timeout

This pattern consists of exactly two messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-on-timeout/".

2.12 In-Optional-On-Timeout

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N
2. An optional message:
 - indicated by a Label component whose {label} is 'In' and {direction} is 'in'
 - received from some node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-optional-on-tiemout/".

2.13 Out-On-Timeout

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'

- sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-on-timeout/".

2.14 Out-Optional-On-Timeout

This pattern consists of exactly three messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N
2. An optional message:
 - indicated by a Label component whose {label} is 'Out' and {direction} is 'out'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-optional-on-timeout/".

2.15 In-On-Correlation

This pattern consists of exactly four messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'In', {direction} is 'in' and {correlation} is 'id'
 - received from some node N
2. A message:
 - indicated by a Label component whose {label} is 'In', {direction} is 'in' and {correlation} is 'id'
 - received from some node N'
3. A message:
 - indicated by a Label component whose {label} is 'Out', {direction} is 'out' and {correlation} is 'id'
 - sent to node N'
4. An message:
 - indicated by a Label component whose {label} is 'Out', {direction} is 'out' and {correlation} is 'id'
 - sent to node N

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/in-on-correlation/".

This pattern might combine the use of the previously defined ones in the exchange among two particular nodes. This pattern might combine the use of the previously defined ones in the exchange among two particular nodes. Further, it can include an arbitrary number of nodes.

2.16 Out-On-Correlation

This pattern consists of exactly four messages as follows:

1. A message:
 - indicated by a Label component whose {label} is 'Out', {direction} is 'out' and {correlation} is 'id'
 - sent to node N'
2. An message:
 - indicated by a Label component whose {label} is 'Out', {direction} is 'out' and {correlation} is 'id'
 - sent to node N
3. A message:
 - indicated by a Label component whose {label} is 'In', {direction} is 'in' and {correlation} is 'id'
 - received from node N
4. A message:
 - indicated by a Label component whose {label} is 'In', {direction} is 'in' and {correlation} is 'id'
 - received from node N'

An operation using this message exchange pattern has a {pattern} property with the value "http://www.deri.org/mep/out-on-correlation/".

This pattern might combine the use of the previously defined ones in the exchange among two particular nodes. Further, it can include an arbitrary number of nodes.

4. MESSAGE PATTERNS IN DECOUPLED HYPERMEDIA SYSTEMS

This section provides an analysis of a number of typical user agent-hypermedia interactions that correspond to several possible scenarios in structural hypermedia. We use the FOHM terminology as embedded in the Auld Leaky link server implementation [11], but the concepts are general enough to be applicable to other similar models. We also assume a scenario of federation of link servers and heterogeneity of user agents. Those scenarios and the corresponding patterns are depicted in Table 2.

Table 2. Behavior for typical scenarios

<i>Pattern</i>	<i>Scenarios</i>
In/Out-resend-on-fault (optional or not)	Basic user agent-link server interactions in which retries are permissible.
Multiple out or in	Queries from the user agent that result in several responses from the same linkserver. This allows the implementation of "asynchronous" delivery of structure.
Ack out or in	Basic user agent-link server interactions in which confirmations are required.
Timeout patterns	Useful for meta-searches in link servers. For example, given a query in the form: <pre><query> <context> <contextvalue key="rating">child</contextvalue></pre>

	<pre> </context> <association missing="variable"> <structure>link</structure> <binding missing="variable" repeatable="true"> <featurevalue key="direction">dest<featurevalue> </binding> </association> </query> </pre> <p>Timeout will be able to disable links coming from a query that is processed by two link servers. The first one will select movies and the second will be used to filter by context those that are not rated for children.</p>
In-On-Correlation	Multiple user agent-link server interactions initiated by the users
Out-On-Correlation	Multiple user agent-link server interactions initiated by the server

5. AN EXAMPLE OF ALIGNING MESSAGE PATTERNS

The structural model presented in this paper and the patterns described have been designed from a general perspective, which allows using them in any application domain. It should be clear by now that such MEPs can only model very simplistic scenarios. However, the structural model allows the combination of various of the MEP in order to define complex message exchanges or conversations as required, being the level of aggregation determined by the necessities of the use case.

In this case, a basic correlation scenario is presented where two users request movie links. The first one, requests links that are not rated for children, while the other demands links rated for 13 year old kids.

```

<query>
  <context>
    <contextvalue
      key="rating">PR13</contextvalue>
    </context>
    <association missing="variable">
      <structure>link</structure>
      <binding missing="variable"
        repeatable="true">
        <featurevalue
          key="direction">dest<featurevalue>
        </binding>
      </association>
    </query>

```

Figure 1. PR13 query

The flexibility of the conceptual model allows encapsulating both queries as one unique element. Thus the structural model is defined as follows:

Elements: <http://www.uah.es/queryChild>,
<http://www.uah.es/queryPR13>,
<http://www.uah.es/results>

Documents: <http://www.uah.es/queryDocument>

Messages: <http://www.uah.es/messageChild>,
<http://www.uah.es/messagePR13>,
<http://www.uah.es/messageResult>

MEP: <http://www.uah.es/In-On-Correlation>

Message Exchange: <http://www.uah.es/correlationLinkServer>

Where, the message exchange *correlationLinkServer* is defined by the MEP *In-Out-On-Correlation* and the messages *messageChild*, *messagePR13* and *messageResult*. Every message

contains always one unique document named *queryDocument*, which in turns holds either one of the elements.

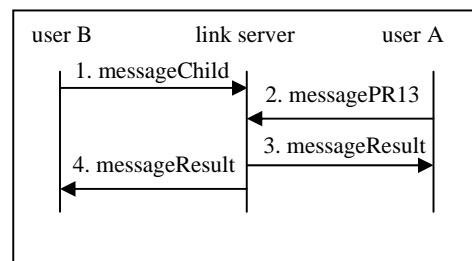


Figure 2. MEP In-Out-On-Correlation

Figure 2 shows the dynamic interaction with the link server.

6. CONCLUSIONS

This paper presented the structural model for SOPHIE and its application to decoupled hypermedia systems. In doing so, the work conducted on [6] has been extended with the addition of a number of patterns that take under consideration some binding-specific information such as faults, acknowledgements, timeouts and correlation among messages. Thus, a broader spectrum of message exchanges is being covered, but always keeping in mind that message exchange patterns identify a minimal contract between parties, and should contain only information that is relevant to the interacting parties.

For the future, the behavioral and operational models need to be defined. Such model should be flexible enough as to readily accommodate different paradigms without impacting the rest of model. The resulting framework should then be applied to extend existing reference models for open hypermedia as FOHM.

7. REFERENCES

- [1] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacs-Nagy, P., Trickovic, I., Zimek, S.: "Web Service Choreography Interface (WSCI) 1.0", <http://www.w3.org/TR/wsci/>, 2002.
- [2] Berners-Lee, T., Hendler, J., and Lassila, O.: "The Semantic Web". Scientific American, 284(5):34-43, 2001.
- [3] Booch, G., Rumbaugh, J., and Jacobs, I.: "The Unified Modelling Language User Guide, Addison-Wesley, 1999.

- [4] Business Process Execution Language for Web Services Java Run Time (BPWS4J), <http://www.alphaworks.ibm.com/tech/bpws4j>, 2002.
- [5] Gudgin, M., Lewis, A. and Schlimmer, J.: "Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns", W3C Working Draft 26 March 2004, <http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326/>.
- [6] Kavantzas, N., Burdett, D., Ritzinger, G.: "Web Services Choreography Description Language Version 1.0", <http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/>, April 2004.
- [7] Roman, D., Scicluna, J., Feier, C., (eds.) Stollberg, M and Fensel, D.: "D14v0.1. On-tology-based Choreography and Orchstration of WSMO Services", <http://www.wsmo.org/TR/d14/v0.1/>, March, 2005
- [8] XML Schema Part 2: Datatypes Second Edition, <http://www.w3.org/TR/xmlschema-2/>. 2004.
- [9] Karousos, Nikos and Pandis, Ippokratis and Reich, Siegfried and Tzagarakis, Manolis (2003) Offering Open Hypermedia Services to the WWW: a step-by-step approach for developers. In *Proceedings International WWW Conference*, Budapest, Hungary.
- [10] Millard, D. (2003) Discussions at the Data Border : From Generalised Hypertext to Structural Computing. *JNCA Special Issue on Structural Computing* 26:pp. 95-114.
- [11] D.T. Michaelides, D.E. Millard, M.J. Weal, and D. DeRoure,(2001). 'Auld leaky: A contextual open hypermedia link server', in *Hypermedia: Openness, Structural Awareness, and Adaptivity* (Proceedings of OHS-7, SC-3 and AH-3), Published in *Lecture Notes in Computer Science*, (LNCS 2266), Springer Verlag, Heidelberg (ISSN 0302-9743), pp. 59-70.