

Discovering Conversations in Web Services Using Semantic Correlation Analysis

Wim De Pauw[†], Robert Hoch[‡], Yi Huang^{*†}
[†]IBM T.J. Watson Research, [‡]Indiana University
{wim, rhoch}@us.ibm.com, yihuan@cs.indiana.edu

Abstract

Composite business applications typically combine characteristics of workflow and transactional applications. In order to handle multiple concurrent sessions and conversations across different nodes and over an extended time period, these applications pass dedicated identifiers inside their messages such as “OrderNumber”. Debugging and understanding these conversations and the flow of conversation identifiers can be challenging in realistic business applications.

We present a new heuristic algorithm to find these conversation identifiers and a method to reverse-engineer the conversations from the content of traced messages. Visualizing the results of this analysis provides a deeper understanding of the data flow, the business process and the interactions between business partners in composite business applications.

1. Introduction

A simple user-facing application, like a stock-quote or a bank-balance inquiry, invokes a service and waits on the response. The network or the middleware layer ensures that the response returns to the invoking client. Multiple requests from a client usually happen in a synchronous fashion. As a result, the behavior of these applications can be observed by following the sequence of messages that make up the control flow. This model can be extended to cover business logic executing in a middleware environment as long as it holds the limitation of executing a session in a single thread.

However more complex business functions are often constructed in a more loosely coupled fashion. Requests can be submitted to modules of the application without waiting on an immediate response. Responsibility for a process may be handed off from one module of the application to another, or to an entirely different application. We refer to applications of this type as composite business applications [1]. In this paper we will call a sequence of messages that is related to a particular business goal a *conversation*.

Where a simple application waits for the completion of a request before continuing, a request in a composite business application may be executed asynchronously,

relying on a call-back mechanism or polling in order to determine its eventual outcome. This more complex pattern of interaction places a greater burden on the application modules to keep track of the state of multiple conversations and to route a message to its correct destination [2]. It is common practice to include application specific data in the messages that are exchanged so that they can carry necessary context from one module to another. The context data will include a data element that serves as a conversation identifier for the application modules that are involved. Recent standards such as WS-addressing [3] promise to let business application developers delegate this task to the middleware level rather than crafting application-specific solutions. Yet, for most existing applications and for composite business applications that span multiple domains of control, the current practice of using application-specific conversation identifiers will continue to be in effect.

The increasing complexity of business applications poses new challenges for understanding application behavior and requires new tools for problem determination. In the absence of standards-based approaches such as WS-addressing, it may be difficult to discover conversation identifiers from trace information, especially without the help of the original developer or documentation. Automatic discovery requires a general approach, independent of the application, to analyze the trace data and to identify likely relationships at run time.

Traditional correlation mechanisms such as ARM [4] provide an API that the application can call to record control flow in a distributed environment. Tools such as the Web Services Navigator [5] or the Eclipse Test & Performance Tools Platform (TPTP) [6] visualize the control flow of messages in composite applications. However, such tools can only combine messages that appear in a direct calling sequence and fail to combine groups of messages that constitute business process conversations. For example, one application may put the result of a transaction in a database, so that it can be picked up by another application. In traditional tools, this would show up as an interruption in the control flow

* Work performed at IBM T.J. Watson Research Center.

where the information is temporarily stored in the database.

In this paper we describe a mechanism for understanding the overall behavior and information flow of a composite application by identifying the conversation identifiers in the content of the messages. We will use the term *semantic correlation* to describe how these content-based identifiers correlate messages across different threads of execution. Moreover, our approach treats the modules of a composite application as black boxes, relying only on capturing the content of messages passed between the modules. Thus, we avoid the problem of expensive instrumentation inside internal components.

2. Finding Identifying Keys

Our algorithm provides an automated analysis for discovering the conversation identifiers in a set of traced messages. It allows us to reassemble sessions and conversations in a composite application. The results from this analysis provide the basis for an interactive visualization of the run-time behavior of a composite application.

Our work has focused on the domain of composite business applications integrated through the use of Service Oriented Architecture (SOA) technologies, in particular those using Web Services to communicate among application modules [7]. As such, we assume that application messages have a structure that can be determined from the content of the messages. Our algorithm relies on the content of the messages in order to find conversation identifiers. It is general enough to find the standard conversation identifiers used in WS-addressing headers, as well as ad-hoc approaches where a conversation identifier is passed inside the message. Since our algorithm uses some simple statistics based on the content of the messages, we assume that we have trace information for a significant number of interactions. In our experiments we processed traces ranging in size between 20 and 3300 messages.

Let us illustrate the key idea with a simple example. Figure 1 shows a set of 100 traced messages of the `orderBook` operation. The figure shows the structure and content for the first message of the set. We assume that the same structure applies to all these messages. We also show the number of unique values or the *cardinality* for each data field for this example. We first find the fields that can be unique identifiers. These are the fields in the messages that have a unique value for each message. Given a large enough number of messages, we can expect `Title` or `ZIP` not to be unique. `OrderNumber`, on the other hand, has as many different values as there are messages. Therefore, `OrderNumber` can serve as a unique identifier for this set of traced `orderBook` messages.

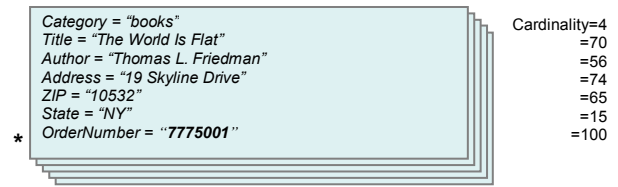


Figure 1. Example of 100 `orderBook` messages with `OrderNumber` as a unique identifier

Let us now consider a second set of messages, `shipBook`, as depicted in Figure 2. Suppose we find a unique identifier `PO` for this set. If, in addition, there is a significant match between the values of `OrderNumber` from the first set and the values of `PO` in the second set, we have identified a semantic correlation between `OrderNumber` and `PO`.

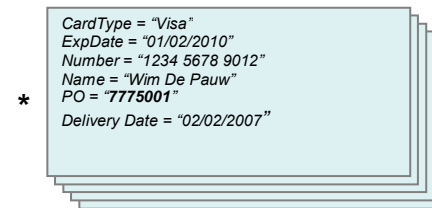


Figure 2. Example of `shipBook` messages where `PO` is unique and matches the values of `OrderNumber`

When we look at these messages in the context of a larger application, we can see how the semantic correlations can link together multiple control flows. The example in Figure 3 shows a configuration with multiple service points (four of them are depicted as boxes). The straight gray arrows indicate the control flow between the service points, gathered with traditional correlation techniques. As suggested in this figure, they only tell us part of the story.

Imagine that we found four data fields with unique values, which are therefore candidate correlation identifiers: `OrderNumber`, `PO`, `CustomerID` and `ShiptoID`. Similar to `OrderNumber` and `PO`, we might also see a match between the values of `CustomerID` and `ShiptoID`. Both matches correspond to two distinct semantic correlations, shown in Figure 3 by the curved arrows between their respective identifiers.

In the case of the semantic correlation between the `orderBook` and `shipBook` messages, their respective data fields `OrderNumber` and `PO` may now serve as correlation identifiers for conversations that include `orderBook` and `shipBook` messages. We can learn more about the causal order between these messages if we look at their respective timestamps. If the timestamp of every `orderBook` message was earlier than the matching

shipBook message, we can infer that in this conversation, orderBook was the cause and shipBook the effect. This causality is shown in Figure 3 by the direction of the curved arrows.

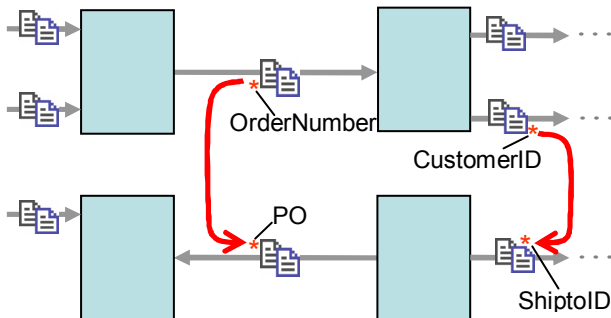


Figure 3. Two semantic correlations shown with curved arrows joining the correlated identifiers

In this example, there is a one-to-one relationship between the values in `OrderNumber` and the values in `PO`. We have extended our algorithm to also find interesting one-to-many and many-to-one semantic correlations between messages. Imagine in our previous example that the ordering of a book, depending on its availability, is followed by one or more of `shippingStatus` messages (not shown in Figure 3). As a result, there may be one or more `shippingStatus` messages with `OrderId = 7775001`, the equivalent of `OrderNumber` in `orderBook`. This implies that `OrderId` in `shippingStatus` is not a unique identifier anymore and that there is no longer a perfect one-to-one match. Nevertheless, cases like this, where only one of the two message schemas has a path with unique values (like `OrderNumber`), may still reveal interesting patterns. The analysis in this case can discover that an `orderBook` message results in one or more `shippingStatus` messages. Conversely, if the same `OrderNumber` was used for multiple books in the same shipping order, we could show a many-to-one relationship.

3. Algorithm

We can discover the conversation identifiers and the semantic correlations following these steps:

3.1. Tracing the messages and their content

We first collect all the messages that are sent between the services under study, including their contents and timestamps. This can be realized with existing technology such as the Web Services Navigator [5].

3.2. Cataloging the message content

Starting from a set of traces of all messages we then build a catalog of these messages partitioned by message type. In the case of a distributed system using Web Services, the messages are encoded in XML and can be grouped by their full message name. In other cases, such as messaging systems, additional metadata like the message channel may be needed to partition the messages by message type.

3.3. Deriving the schemas based on the message contents

For each group of messages with the same message name in the partition we derive a schema based on the content of the messages. In the case of XML, we can derive a schema from the XML paths and the literal values present in the messages. We will use the term *path* to refer to the location of an element or attribute starting from the root of the XML document. We are only concerned with elements and attributes that have content.

3.4. Creating the value tables

We then create a value table for each schema and its associated messages. In a value table each row will represent a message and each column will represent a path of the schema. We now populate each entry in the table with one, multiple or zero data elements for the corresponding message and path, corresponding to the following cases:

- one data element: there is a single data element for the path in the message; for example most cells in table 1 contain one value.
- multiple data elements: there are multiple or composite elements for the path in the message; this may correspond to multiple line items in a document, each having the same path; for example, in table 1, message 1 has multiple elements (`AAA`, `R4`, ...) in the `Discount` path.
- zero data elements: no data element for the path exists in the message; for example, message 3 does not contain any element in the `Discount` path.

Table 1. Value table for orderBook

Msg	Cat.	Title	Author	Address	ZIP	State	OrderN	Discount
1	books	The World i...	Thomas L. Frie..	19 Skyline..	10532	NY	775002	AAA, R4, ..
2	CD	Eyes Open	Snow Patrol	19 Skyline..	10532	NY	342123	Q3
3	CD	My Culture	1 Giant Leap	2 Main Str..	10562	NY	543667	
4	DVD	Himmel übe..	Wim Wenders	54 Jones ..	94133	CA	564356	AAA, RW

In our experiments using reference applications we noticed that a message was sometimes resent with the same content. For example, inquiry functions in the system can generate indistinguishable messages. In cases

where we collected multiple messages with identical content, we used the first occurrence and discarded the rest. Since our analysis is based on content, this seemed to be a reasonable strategy.

3.5. Finding candidate correlation identifiers

In this step we determine which paths in the value tables are good candidates for correlation identifiers. As we demonstrated in section 2, we are interested in pairs of paths that may result in an interesting match. At least one path should have unique values and the second path should have either a set of unique values, or a large enough number of different values. For example, a path with just Boolean values is unlikely to have a meaningful mapping with another path.

For each path p in a schema value table, we take into account two statistics. The first is the total number of data elements or the population for p , Pop_p . Notice that Pop_p can be higher than the number of messages, if this path has entries in its value table with multiple data elements. For example, the path Discount in table 1 is likely to have a Pop_p that is larger than the number of messages or rows in its table. The second statistic is the total number of different values in p or the cardinality of its value set, $Card_p$. Here again, multiple data elements for this path in its value table may produce a cardinality that is higher than the number of messages containing this path. We only consider paths with a large enough number of values, e.g. $Card_p > 2$, screening out frequently occurring, but meaningless, mappings. We now define the indexability α_p of a path p as:

$$\alpha_p = \frac{Card_p}{Pop_p}$$

For example, a path is 100% indexable if each of its data elements has a unique value.

We then define and identify:

1. **Highly indexable paths:** These are paths with an indexability equal or close to 1.0, e.g. $\alpha_p > 0.95$. Allowing a small deviation from 1.0 can identify correlations in cases where there is some noise in the data, such as near duplicate messages.
2. **Mappable paths:** Our goal is to find paths that may have an interesting value match with another, highly indexable path. There are multiple ways to achieve this, and we realized this condition by allowing paths for which none of its values occurs more than γ_{max} times in all of its data elements. For example, a threshold $\gamma_{max}=10$ will prevent that more than 10 data elements in this path with the same value will be matched to a single data element in another highly indexable path. By this definition, a highly indexable path is normally also a mappable path.

This step significantly narrows down the number of paths that are candidate correlation identifiers and restricts the

number of comparisons between value sets of paths later on.

3.6. Finding correlations between candidate correlation identifiers.

We have identified the candidate correlation identifiers in each schema, if they exist, and classified them as highly indexable or mappable paths. We now test pairs of paths from separate schemas to find those pairs that produce a significant match between their value sets. The following steps will identify path pairs with interesting semantic correlations.

1. We first find all the pairs of paths between any two schemas for which: the first path is highly indexable, and the second path is mappable, and a significant overlap exists between the value sets of the first path and the second path.
2. For each such pair of paths found in (1), we match up the individual data elements of the first and the second path. We then assign a causal direction between the two paths based on the timestamps of their respective messages so that we have an origin path and a destination path. We expect to find the timestamps of the messages from the origin path to be all uniformly earlier than the corresponding timestamps in the destination path. If we find conflicting directions beyond the margin of clock accuracy, the path pair may be discarded. Optionally, the pair of paths may be discarded (automatically or after user input) if the number of values found in the origin path is smaller than the number of values in the destination path. Such an observation would violate the concept of causality since we do not expect any new values to appear in the destination path that were not already in the origin path.
3. Each pair of paths found in step (2) can now be categorized as a semantic correlation of the following type:
 - one-to-one: if both the origin and destination paths are highly indexable;
 - one-to-many: if the origin path is highly indexable and the destination path is mappable;
 - many-to-one: if the origin path is mappable and the destination path is highly indexable.

In the context of a business process, this information can reveal: if a given cause has exactly one effect, like the `orderBook.OrderNumber` \rightarrow `shipBook.PO` example in section 2; if one cause has many effects; or if many causes have one effect. We consider a one-to-one semantic correlation stronger than a one-to-many or many-to-one because the value match conditions are more stringent. Notice that this classification of semantic path correlations is orthogonal to the presence of multiple data elements in any of the paths.

After these steps we have discovered the semantic correlations between paths. It tells us which correlation identifiers may be used by the composite application as a common context between different, possibly asynchronous sessions.

3.7. Finding correlations between schemas

We can now define a correlation between two schemas if at least one path correlation exists between the schemas. It is possible that more than one path correlation exists between two schemas. Our objective is to capture those correlations that are most uniquely identifying, therefore we define the correlation type between schemas to be the strongest correlation type of its path correlations. This means that if there is at least one one-to-one path correlation, the schema correlation type will also be one-to-one. The causal direction of a schema correlation is determined by the predominant causal direction from its path correlations.

3.8. Removing transitive semantic correlations between schemas

Transitive semantic correlations may appear among three or more schemas, as shown in Fig 4, ordered by causality. In the example shown, schema A (and its messages) is semantically correlated with schema B (and its messages) by a semantic path correlation $A.x \rightarrow B.y$. Similarly, schemas B and C are semantically correlated by a semantic path correlation $B.y \rightarrow C.z$, and schemas A and C are semantically correlated by a semantic path correlation $A.x \rightarrow C.z$. For visualization purposes, we can assume that the final correlation is redundant information. Therefore, we will not consider it for visualization.

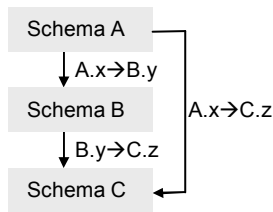


Figure 4. When multiple correlations exist among three or more schemas, we can remove transitive correlations (e.g. $A.x \rightarrow C.z$).

4. Visualizing semantic correlations and conversations in a business process

We used the Web Services Navigator [5] as a visualization environment to add new views showing semantic correlations, correlation identifiers and conversations in business processes. While the existing views in this tool used to show the execution of complex

Web Services applications by focusing primarily on control flow, the new views offer additional understanding into the data flow and the business conversations executed in composite applications.

We illustrate this visualization with an example of a Purchasing application based on the supply chain business interactions defined by the RosettaNet consortium [8]. These interactions define the major steps required to move through the process of requesting quotes, placing purchase orders and fulfilling orders, notifying shipment delays and canceling orders.



Figure 5. Visualization of Purchasing Conversation

Figure 5 shows an excerpt of the execution of this application in a sequence diagram. The two columns represent the different business roles (parties), in this example BUYER and SELLER. Time proceeds in the vertical direction. Control flow is shown, as in most existing tools, as request/response invocations (drawn as horizontal gray arrows) between the parties. In this example we have one `getQuote`, three `purchaseOrder` and two `shipmentNotification` invocations. The `getQuote` and `purchaseOrder` invocations originate from the buyer, the `shipmentNotification` from the seller.

This information alone is insufficient to fully understand the conversation flow in this business application. Therefore, we add the results of our semantic correlation analysis. We draw the message contents for the requests and responses of the invocations as the lighter rectangles. The conversation identifiers are drawn inside as labels in darker rectangles. We also draw the semantic correlations as curved lines, connecting the conversation identifiers. This new visualization first reveals that the quote for merchandise items obtained in `getQuote` was used in three subsequent `purchaseOrders`. This one-to-many semantic correlation is shown as the three curved arrows on the left, connecting the correlation identifiers in each instance. Then, it shows that two of the three `purchaseOrders` were followed by a `shipmentNotification`. These one-to-one semantic correlations are drawn as the two curved arrows on the right.

Now that we learned how these correlation identifiers put the previously disjoint invocations in context, we can better understand the conversations that occurred in this example. The buyer asks a quote from the seller, probably for several items. The seller responds and gives the buyer a `quoteNumber`, (`LEC98...`). The buyer then purchases in three different transactions, three items from the quote list. For each of these transactions, the buyer mentions the `quoteNumber` for the item, to which the seller responds with an `OrderNumber` (`JPMB...`). Whenever an order is ready, the seller sends a `shipmentNotification` to the buyer, mentioning the correct `OrderNumber` (`JPMB...`).

More complex examples might include additional steps in the purchasing process as well as multiple instances of each role. For example a buyer may send out a request for quote to multiple sellers, and then place the order with the lowest-priced seller.

5. Results and performance

We tested our algorithm on a number of non-trivial composite business applications. The first application is the one visualized in the previous section, the Purchasing application. It is based on specifications developed by the RosettaNet consortium and publicly available at [8]. This application is implemented as an event-driven system, relying on asynchronous messages to carry out the purchasing process. The second application, Supply Chain Management, is a reference application of the WS-I Organization; its specification and different implementations are publicly available from [9]. The third application, Service Pac Validation (SPV2) is an order validation application deployed inside IBM. SPV2 relies on BPEL orchestration to selectively call back-end services as part of its validation process. Table 2 shows

some characteristics of each application, a sample run for each and some statistics obtained from our analysis.

Table 2. Summary of testing data

App	Nodes	Msgs	Schemas	Avg paths /schema	Correl. identifiers	Semantic correlations		
						1:1	1:m	m:1
Purchasing	2	790	16	73	24	8	2	2
SCM	8	3300	12	10	4	2	0	0
SPV2	12	220	22	7	2	2	0	0

The second column lists the number of nodes that provided traces from the application being analyzed. The third column, the number of messages that were recorded, gives an idea about the size of the trace. The number of schemas derived from the trace data is shown in the next column. The average number of paths per schema gives an idea about the complexity of the message content for the application. Then, the results of our algorithm are shown as the number of correlation identifiers in the schemas and the total number of semantic path correlations of types one-to-one, one-to-many and many-to-one respectively.

The algorithm and the visualization enabled us to quickly understand how data flowed and which conversations were happening in the business process. For example, in the case of the SCM application, our visualization readily revealed a callback pattern that handles notifications between the Manufacturer and Warehouse services. Similarly, in the case of SPV2, we saw a 1:1 linkage between the initial validation call and a back-end service based on a serial number. In each case, the algorithm correctly selected conversation identifiers, while ignoring spurious linkages based on elements such as time stamps. In general, the accuracy of picking the right identifiers increased with larger volumes of messages.

We also evaluated the performance of our heuristic algorithm in the Web Services Navigator environment to see how well the algorithm handled large traces and how it compared with already existing processing functions in our visualization environment such as loading a trace file. We ran the Purchasing application with progressively larger numbers of messages. We subdivided the processing undertaken into four components: the time taken to load the trace file into the Web Services Navigator; the message parsing; the value table creation and selection; and the correlation of identifiers. Figure 6 shows the time for each of these four components stacked on top of each other. If we take the loading and parsing of the XML messages as a necessary pre-requisite for this type of analysis, we can see that the value table creation and correlation add an additional overhead which is about equal to the pre-processing.

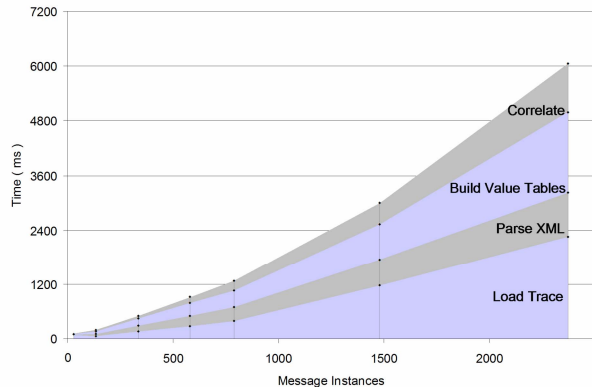


Figure 6. Processing times for increasing number of messages.

In this specific example, when analyzing a trace of 2400 messages consisting of 35MB of data on a 3 GHz Pentium 4 machine, the total processing time was under 6 seconds. Apparently, the total time is dominated by the number of messages in the trace. Overall, in our tests the performance was just slightly worse than linear with the number of messages. In a production environment, the costs of this type of analysis will be dominated by the data collection and pre-processing rather than the semantic correlation. As we discuss in section 7, we plan to look at strategies for minimizing the data collection in future work.

6. Related work

We divide related work into three categories: schema mapping, event correlation and workflow mining.

Schema Mapping: Automated schema mapping among multiple data sources, such as databases, XML messages, has been studied in the context of information integration. Although many algorithms have been proposed for this problem [10,11], fully automated schema mapping is regarded as “AI-complete, that is, as hard as reproducing human intelligence”. Most commercially available schema mapping tools are basically graphical programming tools that let a user to specify schema mappings [12]. Schema mapping algorithms can be categorized in two classes [11]. The first one is schema-level matching and depends on knowing the original schemas of the data sources. Such information may not be completely available from message content traces. A second class uses instance-level matching. These algorithms are usually based on machine-learning or neural network techniques, which try to learn matching information from the training data instances of each schema. All these schema mapping algorithms try to find relationships in data that is static and not moving. Our algorithm, on the other hand, is designed to find correlations in data that is moving between nodes, carried

by messages. It is similar to the instance-level matching since we do not assume knowing message schemas. The dynamic nature of our target domain, messaging systems, allows us to mine the available data more efficiently, for example by considering causality constraints, and translate our findings at a higher level of abstraction. We could however improve the accuracy of our algorithm by taking advantage of some techniques in schema-level mapping, such as thesaurus look-up.

Event Correlations: Correlations of logged events in distributed systems are important for understanding complex systems. A commonly used standard to support distributed event gathering is Application Response Measurement (ARM) [4]. It requires instrumentation of the application(s) under study. Often, middleware has been pre-wired with ARM instrumentation. Other correlation techniques that require significant instrumentation at the system level include Magpie [13] and PinPoint [14]. All these event correlation mechanism that require significant instrumentation, do not provide the higher level business conversations revealed by our algorithm. Aguilera et al. [15,16] propose a method that does not require deep instrumentation and treats the distributed components as black boxes. They infer causal paths from the observed messages by finding nested calls for RPC style communication or by applying signal processing techniques that rely on the time variance between request and response messages. These techniques however may not always work for Web Services, since these and other messaging systems often include asynchronous messaging styles and sometimes have large, unpredictable delays between messages.

Workflow Mining: Workflow mining techniques analyze events and information collected at runtime to discover models of software processes for diagnosis and (re)design [17-19]. These techniques can reconstruct workflow graphs as finite state machines or directed graphs. The algorithms for workflow mining analyze groups of events generated by a workflow. They already assume that each event belongs to a particular ‘case’ (or task). This is not the case with the trace data we collect from Web Services.

7. Future work

We have implemented and described our work in this paper in terms of Web Services. However, the same algorithm can be applied to composite applications using other messaging systems where the messages have structured content. We believe this gives the algorithm potential for broader application in understanding legacy systems integrated through asynchronous messaging.

The current heuristic algorithm can be improved and fine-tuned in a number of ways. We can for example use a sliding time window for the comparison of sets of messages. The algorithm would also benefit from interactive user input to resolve possible ambiguities or to

let the user drill down to a particular piece of the execution. Using a combination of multiple data fields in one message as a correlation identifier (similar to compound keys in database design) would allow the user to discover a wider range of relationships.

If performance and scale become critical, we can split the algorithm in a training phase and an execution phase: first, the training phase would extract the correlation identifiers from a limited, but representative training set; then, the execution phase can efficiently parse and correlate these identifiers across different messages. This could result in a substantial reduction in the amount of data collected from the messages and better performance at analysis time.

In this paper we have focused primarily on the algorithm and we have shown only one type of visualization. In our current research work we are experimenting with different types of interactive visualizations to reveal the correlations between instances of messages as well as patterns of correlations, analogous to execution patterns in Web Services [20]. These visualizations include topology views as well as sequence diagrams.

8. Conclusion

In this paper we have proposed a new heuristic algorithm that can reveal conversations in workflow of composite business applications. It helps developers and business owners to better understand how complex applications with multiple, concurrent, and possibly long lasting sessions, execute. Our semantic correlation complements, rather than replaces, traditional trace correlation methods. However, it provides a deeper understanding of the execution at a higher level of abstraction. Our method is unique because:

- It can show data flow where control flow tracing leaves gaps.
- It does not require any design information of the system under study.
- It analyzes the content of the messages exchanged between the service nodes and does not require expensive internal tracing.

We successfully tried our method on deployed Web Services applications. In all cases we were able to deduce a representative workflow from the results shown in our visualization. Our heuristic algorithm can be applied to any kind of messaging system, as long as the content of its messages is structured and can be parsed. In fact, migrating legacy messaging systems to newer platforms may benefit from the insights gained with our tool.

Acknowledgements

We would like to thank our colleagues at IBM Research including Raju Pavuluri, John Morar, Francisco Curbera and Alex Martens for their feedback and support during the preparation of this paper.

References

- [1] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*, Addison-Wesley, 2004.
- [2] F. Curbera, et al., "The Next Step in Web Services", *CACM* . 6(10), pp 29-24, 2003.
- [3] M. Gudgin, et al., "Web Services Addressing 1.0 - Core," Available from: <http://www.w3.org/TR/ws-addr-core/>
- [4] "Application Response Measurement," Available from: <http://www.opengroup.org/tech/management/arm/>.
- [5] W. De Pauw, et al., "Web Services Navigator: Visualizing the Execution of Web Services," *IBM Systems Journal*, 44, pp. 821-846, 2005.
- [6] "The Eclipse Test & Performance Tools Platform," Available from: <http://www.eclipse.org/tptp/>.
- [7] A. Brown, et al., "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications," Available from <http://www-128.ibm.com/developerworks/rational/library/510.html>
- [8] "RosettaNet standards," Available from: <http://portal.rosettanet.org/cms/sites/RosettaNet/Standards/RStandards/pip/index.html>
- [9] M. Chapman, et al., "Supply Chain Management Sample Application." Available from: <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMArchitecture1.01.pdf>
- [10] P. Brown, et al., "Toward Automated Large-Scale Information Integration and Discovery," in *Data Management in a Connected World*, 2005, pp. 161-180.
- [11] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *VLDB Journal* 10(4), pp. 334-350, 2001.
- [12] S. M. Philip, A. Bernstein, Michalis Petropoulos, Christoph Quix, "Industrial-Strength Schema Matching," *ACM SIGMOD Record* 33(4), pp. 38 - 43, 2004.
- [13] P. Barham, et al., "Using Magpie for Request Extraction and Workload Modeling," Symposium on Operating Systems Design and Implementation, 2004.
- [14] M. Y. Chen, et al., "Path-Based Failure and Evolution Management," *Proceedings of NSDI 2004*, 2004.
- [15] M. K. Aguilera, et al., "Performance Debugging for Distributed Systems of Black Boxes," Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), pp. 74-89, 2003.
- [16] P. Reynolds, et al., "WAP5: Black-box Performance Debugging for Wide-Area Systems," Proceedings of the 15th International Conference on World Wide Web, pp 347-356, 2006.
- [17] W. M. P. v. d. Aalst, et al., "Workflow Mining: a Survey of Issues and Approaches " *Data and Knowledge Engineering*, 47(2), pp 237-267, 2003.
- [18] R. Agrawal, et al., "Mining Process Models from Workflow Logs," Proc. of the Intl. Conf. on Extending Database Technology EDBT'98, pp 469-483, 1998.
- [19] J. E. COOK and A. L. WOLF, "Discovering Models of Software Processes from Event-based Data," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 7, pp 215-249, 1998.
- [20] W. De Pauw, et al., "Execution Patterns for Visualizing Web Services," Proceedings of the 2006 ACM Symposium on Software Visualization, pp 37-45, 2006.