Compatibility Verification for Web Service Choreography

Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer
Imperial College London, 180 Queen's Gate, London SW7 2BZ, UK
{hf1, su2, jnm, jk}@doc.ic.ac.uk

Abstract

In this paper we discuss a model-based approach to verifying process interactions for coordinated web service compositions. The approach uses finite state machine representations of web service orchestrations and assigns semantics to the distributed process interactions. The move towards implementing web service compositions by multiple interested parties as a form of distributed system architecture promotes the ability to support 1) compatibility verification of activities and transactions in all the processes and 2) that the composition is equivalent to the distributed system specification. The described approach is supported by a suite of cooperating tools for specification, formal modeling and providing verification results from orchestrated web service interactions.

1. Introduction

Web Services are an emerging software architecture that harnesses the flexibility and reach of the internet with that of extended distributed systems engineering practices. Web Service workflow languages aim to fulfill the requirement of a coordinated and collaborative service invocation specification to support long running and multi-service transactions. Amongst the key issues in the design and implementation of components in this architecture style for critical business applications, is the formation of compositions (also known as Orchestrations) as a series of interacting workflows [1] and how transactions of activities interact to support the underlying business requirements. This issue is collectively wrapped up in the term Web Service Choreography [2]. These interacting workflows can be constructed using various emerging standards and managed by multiple parties in the domain of interest and as such the task of linking these activities across workflows within this domain is crucial. This work considers the issues in web service choreography by expanding on earlier work discussing modeling local web service compositions [3] which focused on the analysis and verification of behavior models of web service compositions implemented in the Business Process Execution Language for Web Services (BPEL4WS) [4] specification. Additional tool support for this approach has been documented in [5]. In this paper we extend a formal approach to modeling and analyzing the behavior of local web service compositions with that of interacting web service compositions. The aim of this analysis concentrates on the compatibility of processes that take part in the complete composition environment given a system specification. With an accurate analysis process, the implementers of the composition can be assured that interaction will be compatible and therefore match the requirements of interaction. To illustrate how these interacting compositions are verified, we have constructed an example distributed process, consisting of a series of web service workflows. These are translated to a domain independent representation, in the form of the Finite State Process (FSP) notation, which is compiled using the Labeled Transition System Analyzer (LTSA) tool, and properties specified to designate the activities to be assured. The remainder of the paper is organized as follows: Section 2 describes the background and the issues involved in web service compositions illustrated with a motivating example composition based upon an E-Marketplace. Section 3 discusses how compatibility analysis of web service compositions can be undertaken, whilst section 4 details the requirements and steps of an approach to verifying web service compositions, and on the results gained by its use. Section 5 concludes this paper with a broader review of related work, how the described approach is used as part of an automated verification tool and how it is believed that the work described in this paper has contributed to this area of research.

2. Background

Web Services exhibit many similarities to traditional software components that amongst which resemble hosted objects which have a simple, well-defined interface, and that are designed with the expectation of reuse. In fact, the notions and ideas presented in constructing software components are highly applicable to web services, as they could simply be viewed as a type of software component architecture but with the addition of yielding a standard communication model. Some problems of component composition have been reported in [6] as:

- Initially identifying the appropriate components to implement the desired functionality
- determining and resolving gaps between desired functionality and the component's functionality
- specifying the component interactions

These issues are equally applicable to web service components. As web service deployment and use becomes more widespread, the notion of managing the composition of web services to integrate processes together is being highlighted within research and the adopted standards [7]. Web Service compositions, over that of software component compositions however, hold additional issues.

2.1. Web Service Compositions

Web Service Compositions are groups of related services either locally or globally hosted on web servers. The difference with web service compositions over that of traditional software component compositions is that web services compositions focus on the "autonomic open system"; in that they are designed to exhibit a service for varied client use and these can be reused without significant changes incurred to the design, potentially by any interested party. They also exhibit the operational differences where run-time binding is highly loosely coupled enabling a dynamic service invocation. Web Service composition communication tends to be composed through three specific workflow activities (request, receive and reply) [8]. In BPEL4WS requests, receive and replies are specified using the `<invoke>`, `<receive>` and `<reply>` constructs respectively. Figure 1 illustrates a conceptual diagram of communicating partner web service compositions.

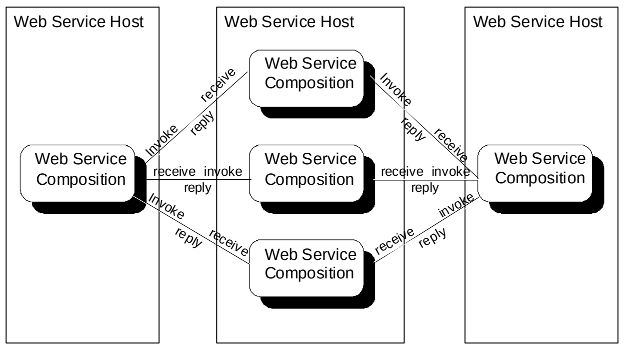

Figure 1. Conceptual diagram of interacting web service compositions

Web Services are also encapsulated components that do not expose their internal behavior and as such, the operation of using them is therefore also somewhat "black box". In additional to the traditional component verification issues, several aspects of web service compositions make a viable approach to verification a highly desirable proposal:

- The coordination of web service composition activities is crucial to synchronizing appropriate behavior between processes. BPEL4WS proposes "abstract processes" for choreography; however this can only currently be validated at runtime.
- The clients of compositions will expect different behavior depending on their individual requests and therefore the composition must be tested against various scenarios to reflect these different sequences of activities.
- There is an assumption that a web service composition will work in any process environment. The assurance can only be given if interacting services know whether a composition exhibits the correct behavior.
- To facilitate coordinated transactions of web service compositions, the appropriate activities must be provided to support web service standards (e.g. WSCI [2], WS-Transaction [9] and WS-Coordination [10] specifications).

Web Service compositions can also be seen as the implementation layer of a multi-stakeholder distributed system (MSDS) [11]. An MSDS is defined as; "a distributed system in which subsets of the nodes are designed, owned, or operated by distinct stakeholders. The nodes of the system may, therefore, be designed or operated in ignorance of one another, or with different, possibly conflicting goals". With a service-oriented architecture the focus is on interaction with multiple parties and the behavior could be somewhat ad-hoc depending on the requirements of the partner services. The desirable element here is that to concisely reason about the applicability of a solution we must be able to determine if a distributed solution is correctly orchestrated. Properties to satisfy this verification may consist of a series of questions about the composition; for example; if a request to purchase a product is sent to a partner process, will the process eventually confirm the purchase?

2.2. Motivating Example

A simple but effective example of the issues in web service compositions is described given an E-Commerce

Marketplace system. One such scenario in this system could include the seller and buyer interaction of products with the Marketplace. We have illustrated such a process as a message sequence chart composition in Figure 2. The example consists of four phases, from initial requests of offer and buy, through negotiation of prices, to a price agreement being made.

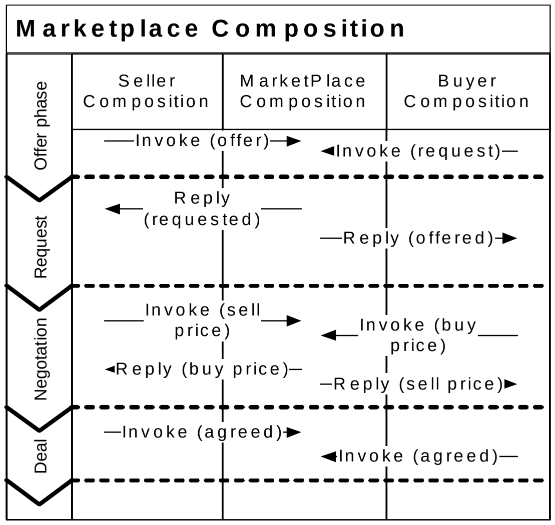

Figure 2. An example scenario of interacting web service compositions

This composition consists of three workflow processes. From the Seller composition perspective, the behavior of the marketplace of interest is that in the ability to receive a request for "offering a product", "offers sell price" and "deal made". However, from the buyer perspective the behavior of interest is the ability to make requests for "request for product", "request price" and "deal made". Although both clients of the marketplace are ultimately interested in whether deal can be made, the paths to this end are different. For this paper, we concentrate on the perspective of the Seller, Buyer and Marketplace compositions and their initial interaction behaviors in this example.

3. Compatibility of Compositions

Three levels of compatibility for component compositions have been previously reported in [12]. These are defined as *interface*, *behavior* and *input-output (data)* compatibility. Whilst input-output compatibility is of interest, it is not the main focus of this paper. Therefore, for behavior we apply the first two of these concepts for compatibility against web service compositions under the following topics;

- **Interface Compatibility** – focusing on semantics of correlating invocations against receiving and message replies between partner processes.
- **Safety Compatibility** – assurance that the composition is deadlock free and is checked against partial correctness of transitions.
- **Liveness Compatibility** – assurance against starvation of progress (that the service process eventually terminates) and that messages received are served on a first-come-first-served basis.

Safety Compatibility focuses on performing safety analysis of composition models, and determining if there exist any deadlocks or other properties specified by the analyst. Liveness Compatibility focuses on performing progress analysis of composition models, and determining if the composition exhibits behavior which does not meet requirements of success. An example of this is that the marketplace process can finally provide a "dealmade" result. As a pre-requisite for specifying safety or progress analysis, we must be able to observe the path to this observable result which relies on the core activity mapping (interface compatibility). The focus of verification for *safety* and *liveness* therefore is from the perspective of the behavior of each client process, in that, for each verification run we are interested in whether a process is activity compatible with its partner compositions. As the verification process iterates through all client requirements for assurances of process, by changing focus of process compatibility, an entire interaction assurance can be given.

As part of the BPEL4WS specification, abstract processes can be defined which hide the private implementation of interactions within the process. These are not directly executable, but they can indirectly impose behavior compliance upon private processes executed by the BPEL4WS orchestration server. Abstract processes may assist in execution however, as a BPEL4WS orchestration server validates and assures public protocol conformance of executing processes. Whilst abstract processes assist in this way, we scope this approach to the design of the core (known as private) process to capture the real interaction and goals of each BPEL4WS implementation and thus aid in complete distributed system implementation verification. We discuss further work around the abstract process later. The core semantics of BPEL4WS, discussed in [3, 8, 13] describe how the language provides interactions of web service compositions. To model these semantics in our approach requires us to map between activities in processes, and to evaluate the questions suggested previously, and that the order of these activities is sufficient to provide the required composition behavior.

To illustrate the semantics of this interaction, we discuss how these interactions occur in BPEL4WS composition examples.

4. Verification Approach

In this part of the paper we describe the requirements and corresponding steps of an approach, for verifying the compatibility of web service compositions specified in the BPEL4WS notation, illustrated by examples.

4.1. Requirements for Verifying Compatibility

The requirements for our approach are detailed as a series of steps; from preparation to performing the compatibility checks. These steps are:

- Gather all orchestrations to be used in the composition compatibility verification
- Analyze composition orchestrations to gather all activities for each process
- Model activities based upon semantics of operation (BPEL4WS construct semantics) between processes and partners
- Translate BPEL4WS composition into a model representation (FSP)
- Compile processes into finite state machine and compose all processes into a Labeled Transition System
- Perform Safety and Progress Checks on composed processes to verify that safety properties are not violated and to analyze that compatibility requirements hold true in the composition using formal modeling verification techniques.

To commence the approach, illustrated in Figure 3, the implementers must collate the processes that are required as part of the verification specification. In our example, these are the Seller, Buyer and Marketplace compositions. The activity analysis phase determines which activities the composition is performing (e.g. gather all request, receive and reply activities). These activities are then mapped between the compositions, and semantics placed on how one invoke is matched with a receive statement for which we describe the process later. A model is then produced by translation of the BPEL4WS and process semantics applied. When this translation has completed, the resulting processes can be composed together to form a complete model of the composition and checks can be made on the model to ascertain whether success on compatibility of behavior is achievable.

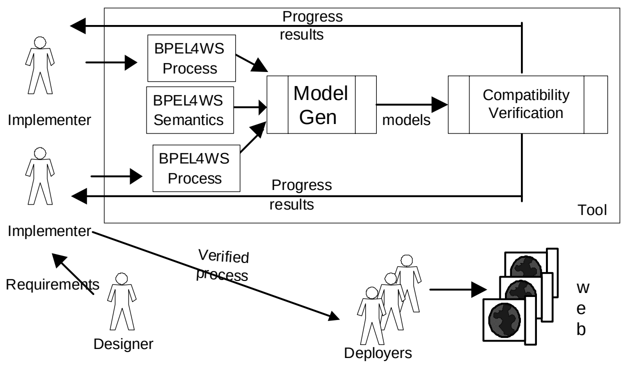

Figure 3. Compatibility Verification Approach

4.2. BPEL4WS Composition Examples

In the Marketplace example we have the following partial BPEL4WS construction for the Seller interaction flow of invocations. The flow of two invocations, requests the marketplace service twice (concurrently) to sell different items.

```
<flow name="SellerFlow">
 <invoke partner="marketplace" portType="sellPT"
   operation="Offer"
   inputVariable ="sellinfo"
   outputVariable="offered" />
 <invoke partner="marketplace" portType="sellPT"
   operation="Offer"
   inputVariable ="sellinfo2"
   outputVariable="offered2" />
  <!— further activities here…. >
</flow>
```

Equally, for any offered items, we require a buyer process to support buyers requesting items;

```
<sequence name="BuyerSequence">
   <invoke partner="marketplace" portType="buyPT"
     operation="Request"
     inputVariable ="buyinfo"
     outputVariable="offereditems" />
  </sequence>
```

Whilst a corresponding activity in the Marketplace interaction process between seller and buyer, is:

```
<sequence name="RequestPhase">
  <receive partner="seller" portType="tns:sellPT"
    operation="Offer" variable ="sellitem"
    createInstance="yes" name="SellerReceive" />
  <receive partner="buyer" portType="tns:buyPT"
    operation="Request" variable ="reqitem"
    createInstance="yes" name="BuyerReceive" />
  <reply partner="seller" portType="tns:sellPT"
    operation="Offer" variable ="buyitem"
    name="SellReply" />
</sequence>
```

The potential "cause effect" of the composition interaction of seller, buyer and marketplace processes is illustrated as a message sequence chart in Figure 4.

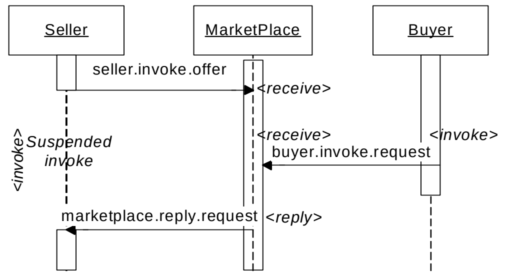

Figure 4. Synchronization of related activities in Marketplace Composition

4.3. Semantics of BPEL4WS communication

A detailed translation of BPEL4WS to FSP models is given in [3], however, we add to this the semantics for how to translate the connectivity and communication between activities of the partner processes rather than from a single process focus. To commence this we require a process to analyze which activities are partnered in the compositions. For example, invoke from the seller process (to offer a product) will be received by the marketplace process (receive offer product from seller). Equally the buyer invokes activity, to request for a product, will be aligned with receive in the marketplace process. In BPEL4WS, the communication is based upon a protocol of behavior for a local service. However, the partner communication can concisely be modeled using the synchronous message passing model, described in [14]. The Sender-Receiver example discussed uses *Channels* to facilitate message passing between such a sender and receiver model. The representation of a channel in BPEL4WS is known as a *port*. The significant element of this discussion used in our process is that of synchronization of the invoking and receiving messages within compositions between ports and whether this has been constructed concurrently (FLOW construct in BPEL4WS) or as a sequence (SEQUENCE construct in BPEL4WS) of activities.

4.4. Partner Activity Models and Semantics

The process models for the example can be constructed in FSP by translation of flow and other activity constructs in each process to corresponding FSP statements and then composed as one complete model. Transition semantics are labeled using the construct name (invoke or receive), *partner* (seller) name, partner process name (marketplace) and by the *operation* being requested (e.g. offer a product). These provide us with a set of labeled process transitions, such as "*invoke_seller _marketplace_offer*". If there is more than one invoke in the seller process, then this can be sequentially numbered. Equally, the message receive activity in the Marketplace process example gives as an example translation of "*receive_seller_marketplace_offer*". The labeled transitions can then be synchronized together by searching for the relevant receive activity given an invoke transition. For example, the transition of *invoke_seller_marketplace_offer* is followed by a *receive.seller.marketplace_offer* in the partner process. To illustrate this activity, the examples previously in BPEL4WS are listed below as FSP process models (Actions -> Process, with || denoting a parallel composition and ; a process sequence).

```
// SELLER FLOW MODEL
SI1 = (invoke_seller_marketplace_offer->END).
SR1 = (invoke_seller_marketplace_reply->END).
SI2 = (invoke_seller_marketplace_offer2->END).
SR2 = (invoke_seller_marketplace_reply2->END).
SSEQ1 = SI1; SR1; END.
SSEQ2 = SI2; SR2; END.
||S_BPELModel = (SSEQ1 || SSEQ2).

// BUYER SEQUENCE MODEL
BI1 =(invoke_buyer_marketplace_request->END).
BR1 =(invoke_buyer_marketplace_reply->END).
BSEQ1 = BI1; BR1; END.
||B_BPELModel = (BSEQ1).

// MARKETPLACE SEQUENCE MODEL
MPSI1=(receive_seller_marketplace_offer-> END).
MPSR1=(reply_seller_marketplace_offer->END).
```

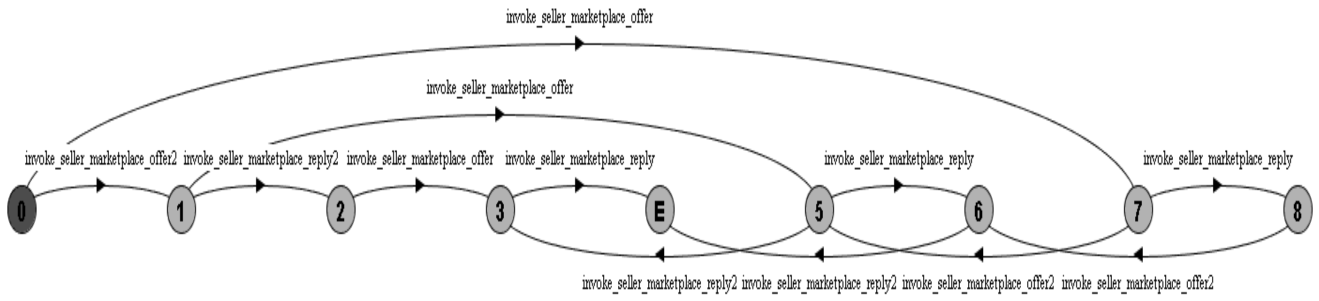
invoke_seller_marketplace_offer
invoke_seller_marketplace_offer
invoke_seller_marketplace_offer2
invoke_seller_marketplace_reply2
invoke_seller_marketplace_offer
invoke_seller_marketplace_reply
invoke_seller_marketplace_reply
invoke_seller_marketplace_reply
0
1
2
3
E
5
6
7
8
invoke_seller_marketplace_reply2
invoke_seller_marketplace_reply2
invoke_seller_marketplace_offer2
invoke_seller_marketplace_offer2

Figure 5. Graphical LTS of Seller BPEL4WS Process

```
MPBR1=(reply_buyer_mp_request ->
      reply_seller_mp_offer -> END).
MPSSEQ1 = MPSI1; MPSR1; END.
MPBSEQ1 = MPBI1; MPBI1; END.
||MP_BPELModel = (MPSSEQ1 || MPBSEQ1).
// MARKETPLACE SEQUENCE END
```

Each FSP Process can be compiled as a Labelled Transition System (LTS), for which an example for the seller flow process is illustrated in Figure 5. However, we need to be able to connect models for verification.

4.5. Message Invocation Connectors

To model the interaction between processes we require a process link between the <invoke>, <receive> and <reply> actions of the BPEL4WS processes and a model of how these interactions are buffered across the distributed system. Partner process activity interactions can be represented in FSP by using the notion of a *connector* (Figure 6), which encapsulates the interaction between the components of architecture. This is implemented in FSP as a *monitor*, allowing us to combine the concepts of information hiding and synchronization (Figure 7).

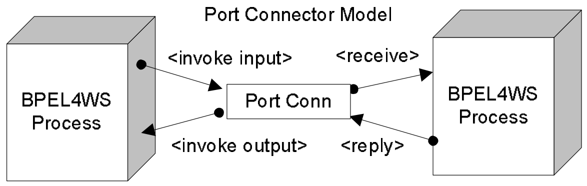

Figure 6. BPEL4WS Process Port Connector

"Rendezvous" (Request-Reply) invocations are specified in BPEL4WS with the <invoke> construct, with both *input* and *output* container attributes. To model these types of interactions, we use a generic synchronous port model for each process port. In FSP, an example of this is specified as follows;

```
// SYNCH REQUEST-REPLY PORT MODEL
PORT_REPLY = (reply[v:M]->
invoke_output[v]->PORT_REPLY).
PORT_INVOKE = (invoke_input[v:M] ->
receive[v]-> PORT_INVOKE).
||PORT1 = (PORT_INVOKE || PORT_REPLY).
```

Synchronous invocations specified with the <invoke> construct and only an *input* container attribute declare an interaction on a request only basis (there is no reply expected). The model for this is;

```
// SYNCH REQUEST ONLY PORT MODEL
PORT_INVOKE = (invoke_input[v:M] ->
receive[v]-> PORT_INVOKE).
||PORT2 = (PORT_INVOKE).
```

With both of these invocation model types, the connection interaction for invoke activities in BPEL4WS can be modeled effectively using transition links for send, receive and reply processes in FSP. The task of modeling the invocation process and port is completed by using the re-labeling feature of FSP linking the appropriate activities between process and port. An example of this is;

```
/ {invoke_seller_marketplace_offer /
invoke_input,receive_seller_marketplace
_offer / receive}
```

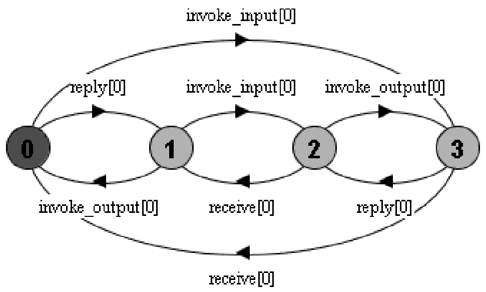

Figure 7. Connector Model for Invoke-Reply

4.6. Parallel Composition with Connectors

To complete the modeling of the compositions, we specify an architecture model composing the previous models for seller, seller port, marketplace, buyer port and buyer processes. The statement for this is;

```
||CompArch = (S_BPELModel || S_PORT1 ||
MP_BPELModel || B_PORT1 || B_BPELModel).
```

Using the LTSA tool [14], we are now able to compile these connectors as FSP models into a complete LTS, and begin to assess correctness. A partial view of a compiled architecture LTS model for the seller interactions with seller port and the Marketplace process is illustrated in Figure 8. As we now have a complete model, we can use this as the basis to perform our verification tests.

4.7. Safety and Liveness Property Verification

A safety property *Q* in FSP is represented by an image of the LTS of the process expression that defines the property. The image LTS has each state of the original LTS and has a transition from each state for every action in the process alphabet of the original. Transactions added to the image LTS are to the *error-state* and signify a failure in verification. Formally, to check a safety property of an LTS we use the following formula;

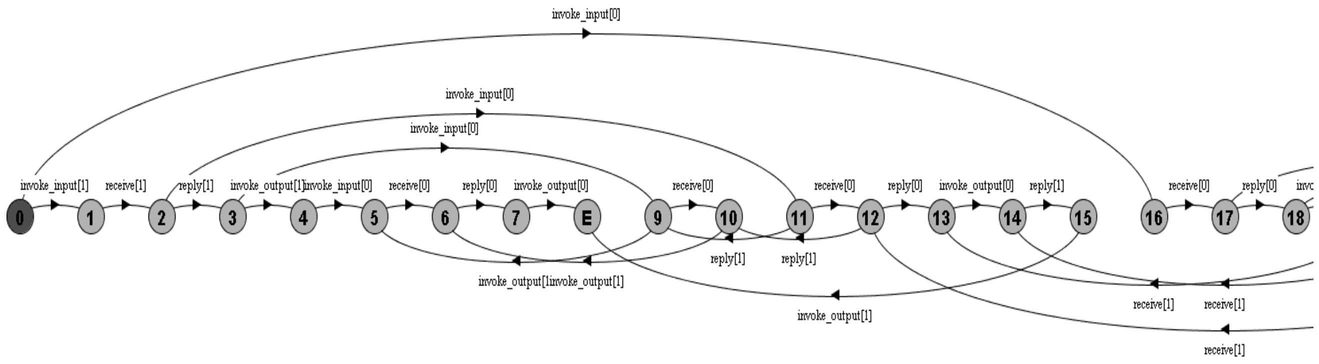

Figure 8. Partial View of Graphical LTS model of Seller-Marketplace composition port interactions

Property Q = E : lts(Q) = $_{\text{def}}$ image(lts(E)),
For an LTS P = <*S, A, Δ, p*>, image(P) =
<S ∪{π}, A, Δ', q>, where *Δ' = Δ ∪{(s,a, π)}*|*s ∈ S ,a ∈ A, and s' ∈ S : (s,a,s') ∈ Δ*
where S is a finite set of states, A denotes the alphabet of P and Δ a transition map between actions

The result for either safety or liveness verification tests is true whenever a required property (*Q)* for verification is observed in (*process*$_1$ || *process*$_{2\,...}$|| *process*$_x$) where || is an FSP parallel process composition operator. The main function of verifying safety compatibility between compositions is that it is deadlock free. Deadlock analysis of a LTS model involves performing an exhaustive search of the LTS for deadlock states (i.e. states with no outgoing transitions. This search is built into the LTSA tool which when invoked can detect such a presence. The LTSA supports Compositional RΔeachability Analysis (CRA) of a software system based on its architecture. CRA incrementally computes and abstracts the behavior of composite components based on behavior of their immediate children. Given the model of the composition and a set of properties (assumptions) we are interested to verify, we can also use the LTSA tool to perform a series of traces on this set of safety properties. For example, the following property verifies that in a given composition, the seller process invocation of the "*offerproduct*" operation has an equivalent receive operation in the marketplace process.

```
property SELLERINVRECEIVE =
  (invoke_seller_marketplace_offer->
  receive_seller_marketplace_offer).
```

Similarly, we can specify an FSP progress statement to test liveness properties with an expression of a required test that a sequence will eventually occur. Using our example previously, to test that a reply can finally be given we could express a property as;

```
progress MPSELLERINVREPLY =
  {invoke_seller_marketplace_reply}.
```

We have also been able to model fault tolerance and compensation sequences in BPEL4WS processes, however, further work is required on verification in this area to manage the scoping and specific fault occurrences within process activity groups.

4.8. Analysis of Example Composition

We illustrate our approach with an analysis of the example used in section 4.2. We perform the translation of BPEL4WS to FSP, and then use the LTSA tool to compile the models and perform safety and progress checks for each compatibility test. In this case, no deadlocks are found. However, if we introduce a specific error, such as omitting a reply to the seller process from the Marketplace process, the following safety check results are displayed;

```
Trace to DEADLOCK:
  invoke_seller_marketplace_offer.0
  invoke_seller_marketplace_offer.1
  receive_seller_marketplace_offer.0
  invoke_buyer_marketplace.request.0
  receive_buyer_marketplace_request.0
  reply_buyer_marketplace_request.0
  invoke_buyer_marketplace_reply.0
  *deadlock-seller reply not observed*
```

The deadlock appears due to the lack of a reply for a seller in the marketplace process. By iteratively building the BPEL4WS processes, performing the translation and verification approach described in this paper, the implementers can be assured that an interacting coordinated set of compositions can be deployed and executed appropriately. Whilst this is a time consuming task by hand, the use of an automated verification tool for this approach provides a simplified task to include in the wider development process.

5. Discussion and Conclusions

Other work on modeling and verification of web service compositions have been reported in [15]. This work was reported using an earlier, non-standard, composition language and translated into Promela (automata) processes. The Web Service Choreography Interface (WSCI) [2] also describes the interaction of web service compositions; however, implementation verification from a business system composed of web service orchestrations will need to be thorough.

To automate this approach we are leveraging the work for local web service composition verification previously, and adding multiple process verification to the feature list. The functionality is currently a plug-in [5] for the LTSA tool, however, we will be migrating this tool across to the open-source Eclipse development environment for distribution to a wider audience and to seek to integrate with other web service design tools.

We have presented an approach to modeling and verifying web service compositions constructed in the BPEL4WS emerging standard specification. The use of our approach is currently limited to the availability of private processes implemented. BPEL4WS defines an abstract process which furthers the interaction interface using asynchronous "callback" declarations of ports. We are seeking to broaden this work by combining both abstract and private models and providing full verification for choreography at implementation and coordination layers. Further work is also required with respect to transactional modeling and to verify this against compensation routines between processes, as related faults must explicitly be mapped to known fault naming conventions. We believe that in parallel with work such as WSCI for choreography, WS-Transaction for transaction handling and WS-Coordination/WS-Policy for profiling, roles and service assurance during web service conversations there is sufficient need for a verification process to confirm that designs and implementations will prove successful once deployed. By combining these models with the complete system model, the additional verification of these routines is incorporated into our approach. We are also seeking to provide specification decomposition based upon a resource model of how BPEL4WS processes are composed to distributed requirements.

6. Acknowledgements

The authors would like to acknowledge that this research was supported, in part, by the STATUS ESPIRIT project (IST-2001-32298), the EPSRC READS project (GR/S03270/01) and by an IBM Innovation Award (2004).

7. References

[1] D. Chakraborty and A. Joshi, "Dynamic Service Composition: State-of-the-Art and Research Directions," University Of Maryland, Baltimore, December 19 2001.

[2] A. Airkin, S. Askary, S. Fordin, W. Jekeli,, D. Orchard, K. Riemer, "Web Service Choreography Interface (WSCI) 1.0," W3C Working Group 2002.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions," presented at Eighteenth IEEE International Conference on Automated Software Engineering (ASE), Montreal, Canada, 2003.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, S. Thatte, and S. Weerawarana, "Business Process Execution Language For Web Services, Version 1.1," May, 2003.

[5] H. Foster, "LTSA-BPEL4WS Tool." Department of Computing, Imperial College London, 2003. Available to download at http://www.doc.ic.ac.uk/ltsa/bpel4ws.

[6] M. Fowler, "Components and the World Of Chaos," *IEEE Software*, vol. 3, pp. 83-85, 2003.

[7] J. Yang and M. P. Papazoglou, "Service Components for Managing the Life-Cycle of Service Compositions," *Information Systems*, 2003.

[8] W. M. P. v. d. Aalst, M. Dumas, and A. H. M. t. Hofstede, "Web Service Composition Languages: Old Wine in New Bottles?," presented at Proceeding of the 29th EUROMICRO Conference, Los Alamitos, CA, 2003.

[9] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte, "Web Services Transaction (WS-Transaction)," BEA Systems, IBM, Microsoft Corporation, Inc 2002.

[10] F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey, "Web Services Coordination (WS-Coordination)," BEA Systems, IBM, Microsoft Corporation, Inc 2002.

[11] R. J. Hall, "Open Modeling in Multi-stakeholder Distributed Systems" presented at Proc. First Workshop on the State of the Art in Automated Software Engineering,, 2003.

[12] M. Larrson and I. Crnkovic, "New Challenges for Configuration Management," presented at the SCM-9 workshop, Toulouse, France, 1999.

[13] R. Khalaf, N. Mukhi, and S. Weerawarana, "Service-Oriented Composition in BPEL4WS," presented at The Twelfth International World Wide Web Conference, Budapest, HUNGARY, 2003.

[14] J. Magee and J. Kramer, *Concurrency - State Models and Java Programs*: John Wiley, 1999.

[15] S. Nakajima, "Model-Checking Verification for Reliable Web Service," presented at OOPSLA 2002 Workshop on Object-Oriented Web Services, Seattle, Washington, 2002.