

Dynamic Protocol Aggregation and Adaptation for Service-Oriented Computing

Linh Duy Pham, Alan Colman, Jean-Guy Schneider
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Victoria, Australia
{lpham, acolman, jschneider}@ict.swin.edu.au

Abstract

Service Oriented Computing (SOC) is a paradigm for building new software applications from existing loosely-coupled services. During service composition, services available to play different roles in a composition may have variations in their business-level protocols. These protocols may involve communication between two services in a point-to-point relationship, or communication among more than two services. Furthermore, as the business processes change, those protocols need to be modified to reflect the changes. In this paper, we propose a method to describe protocols between roles that services will play in the composition by specifying the temporal constraints. An automated aggregation of those protocols is then carried out to produce role-centric views. Protocol compatibility of available services can then be checked against these views. We will show how our approach supports the incremental specification of protocols and the flexibility of changing protocols.

1. Introduction

In a business transaction, different services provided by business applications need to interact with each other in order to accomplish a predefined goal. Service-Oriented Computing (SOC) and networking infrastructure enable developers to build distributed applications which comprise of loosely-coupled services from different businesses. SOC facilitates inter-organisation interactions, requiring less human intervention than earlier approaches [1]. In SOC, application development is a process consisting of service discovery, evaluation and composition. To support composition of services, a variety of standards

such as WSDL and SOAP have helped resolving the heterogeneity in implementation platforms. However, standardisations in the syntactic interface description of services (e.g. WSDL) alone is not sufficient to ensure the correct interoperation of services [2]. Previous work in component-based software engineering suggests that there are 4 levels of component interface specification: *syntactic, behavioural, synchronisation* and *QoS (Quality of Service)* [3-5]. In this paper, we are focussing on behavioural interoperability, in particular the sequence of exchanged messages (protocols) between services. The protocols need to be encoded and enforced so that the services are invoked correctly. We need to specify protocols within a compositional representation that is independent from the implementation of particular services.

The description of point-to-point protocols between services in a composition is necessary but may not be enough to ensure that a business transaction is carried out correctly. It may be necessary to model the protocol interactions between multiple (3 or more) services in a composition. Consider, for example, a holiday booking scenario. While a hotel room might be initially reserved, the finalisation of the hotel booking can only be done after the booking of air tickets is confirmed. There is therefore an interaction between the client-to-hotel booking protocol and the client-to-airline booking protocol. This requires dependencies *between* protocols to be specified and enforced. In an orchestrated composition, these dependencies are typically captured in the sequencing of scripted service invocations (e.g. as in BPEL). However, in the ever-changing world of Web services, this imperative approach is brittle if the protocols, which are implicitly followed in the orchestration script, need to change. Protocols may change as the business processes change to better suit the customer's needs. Protocols

may also change as various services with different protocol requirements are bound to the composition. Consequently, the service composition has to update its protocols and their dependencies to reflect such changes. To facilitate this process, the service composition should have a mechanism that enables it to adapt to protocol changes dynamically. The composition also needs to present an aggregated view of the protocols and their dependencies relevant to each of the services in the composition.

In this paper, we address the problem of (i) how to represent protocols between services, (ii) how to aggregate the protocols and their dependencies to create consistent views, (iii) how to make sure protocols used by the services are compatible, and (iv) how to enforce the agreed protocols. The approach taken in this paper is to maintain a run-time organisational representation of the service composition using the ROAD (Role Oriented Adaptive Design) framework [6, 7]. This representation is internal to the compositional infrastructure, which includes entities that represent abstract service definitions (*roles*) and *contracts* between those roles. In ROAD, a role is a “position” within an organisational structure that can be filled at different times by different services that provide the same functionalities.

We will show how the definitions of valid protocols between services are stored in contracts in terms of temporal constraints specified by using the Interaction Rule Specification (IRS) [8], an extension of Specification Pattern System (SPS) [9]. As a service may be involved in a multi-party interaction, it follows that the role that the service plays may have associations (contracts) with a number of other roles. The dependencies between protocols in a multi-party interaction are modelled as dependencies between protocols specified in contracts. For a service to play a role, it must be able to follow the protocols prescribed by that role and the dependencies between contracts associated with that role. We therefore need to be able to describe a set of aggregated 'role-centric' protocols that services can follow. We will show how multi-party protocols can be aggregated and reasoned about within the composition using the IRS formalism.

The rest of this paper is structured as follows. Section 2 presents a motivating example as a basis for further discussion. Section 3 gives a brief introduction to the ROAD framework. We discuss our approach to protocol specification in Section 4. Section 5 illustrates how the aggregation of protocols and the consistency checks are done. Then the compatibility checking and run-time monitoring of protocols are detailed in Section 6. In Section 7, we show that our approach to

protocol specification supports the dynamic changes of protocols. We review some related work in protocol specification in section 8. Finally, section 9 concludes with a summary and a discussion of future work.

2. A motivating example

In this section, we present a scenario of a book supply chain service composition to illustrate the differences in protocols between interacting services and the need to have an aggregated specification of protocols. In the subsequent sections, we will detail our approach based on this scenario.

A large institutional library frequently buys books from different book vendors. In order to automate the process, the library decides to introduce a Web service (Buyer role) that will take an order from a librarian and interact with various vendors' Web services (Vendor roles), e.g. Amazon's Web service, to ask for quotes and order books. Clearly, there are many business level protocols that govern the interaction between the library and the vendors. Some of these involve the sequences of interactions, for example terms of payment (payment before delivery, or delivery before payment), conditions of order cancellation, non-delivery of goods, etc. As these services were developed by different organisations, there may be mismatches in the constraints on the sequence of interactions between services (i.e. “*protocol mismatch*” [10]) due to different assumptions made during development process [10-12]. Therefore flexibility and adaptation in protocols are needed in order for the library's Web service to interact with a range of different book vendors' services that have not been foreseen during the design phase. Figure 1 illustrates a point-to-point protocol mismatch between Buyer and Vendor services. If the library requires a book delivery before the payment can be made, while the vendor requires payment before it can deliver the books, then they cannot carry out the transaction although the vendor has good prices on the books required.

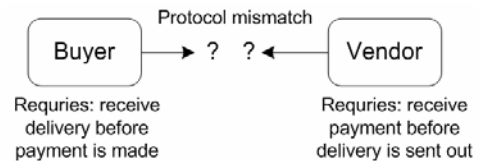


Figure 1: Point-to-point protocol mismatch

In addition, as pointed out in the previous section, protocol mismatches may also arise because of dependencies *between* protocols in a composition. Consider a scenario in which three services are

time in order to verify the interaction. In particular, contracts define *protocol clauses* that describe permissible sequences of transactions that can occur between roles.

In the ROAD framework, contracts are implemented using *association-aspects* [13, 14], an extension to the AspectJ compiler. Such association-aspects allow contract instances to be created that associate groups of objects (in this case associations of role instances). ROAD contracts (being association aspects) have the ability to intercept the communications between these roles using *pointcut* pattern matching on method signatures. This interception is used to verify and prevent communication not authorised by the contract. In this way contracts perform a similar function to *interceptors* in conventional middleware. As specified in our previous work [6, 7], ROAD contracts have the functions of Interaction Monitor (monitoring the messages sent and received between roles) and Request-Response Transaction Monitor (monitoring and associating between request and response messages). In this paper, we extend its function to include the Protocol Monitor which maintains the state of the conversations (sequences of transactions) between the roles involved in the contract and checks if they follow the predefined protocols.

An *organiser* provides an overall management over roles and contracts within its composition. Organiser can create (and destroy) roles, and create (and revoke) contracts between these roles and the binding between roles and services. The organiser also provides a management interface that allows the non-functional requirements of the composite to be set. These functionalities are described elsewhere [7]. In this paper, we will focus on the organiser's management of protocols. In the context of maintaining valid protocol descriptions in its composite, the organiser is responsible for writing protocols to the contracts it creates between the roles. It maintains a model of the dependency constraints between these contract protocols. As shown in Figure 3, the organiser also is responsible for aggregating the protocols in a role's various contracts into a single role-centric protocol description for each of its roles (Cf. Section 5). In the example given in Figure 3, this aggregation is trivial for the Vendor and Buyer roles because each of these roles is only involved in a single contract. However, the Broker role has two contracts – one with the Buyer and one with the Vendor. We describe the formalism for specifying protocols in contracts and performing this aggregation in Section 4.

The adaptability that ROAD provides is the ability to have different services playing roles at different

times, to create new role instances and to revoke/create contracts between roles thus the structure of a composition can be changed dynamically. In this paper, we extend the ROAD framework so that it can cope with changing business protocols dynamically by providing protocol compatibility check, monitoring and enforcing interaction protocols at run-time. The ROAD framework only provides an infrastructure to maintain the representation and enforcement protocols, the knowledge required to define protocols and their dependencies is provided by the business analysts and programmers.

4. Protocol specifications and protocol dependencies

In order to specify the protocols, we use *temporal constraint* formalism. We adopt the Interaction Rule Specification (IRS) [8], which is an extension of Specification Pattern System (SPS), to specify the temporal constraints of the protocols between two roles and store the specifications in the contract.

The SPS provides a high-level specification abstraction to specify temporal properties of system interactions [9]. This temporal approach has some advantages over other formalisms (e.g. π -calculus, Petri nets) in that we only need to define the order of occurrence or absence of the events of interest. The SPS allows us to specify the protocols in terms of *occurrence patterns* (the occurrence of a given event/state during system execution) and *order patterns* (the relative order in which multiple events/states occur during system execution). Each pattern has a *scope*, which is the extent of the program execution over which the pattern must hold. There are five basic kinds of scopes: global, before, after, between, and after-until [15]. It supports incremental description of system properties in an easy to understand manner compared to other formalisms [2, 8, 9].

Figure 4 shows the protocols in Buyer-Broker (BB) contract using SPS in the book procurement scenario as depicted in Figure 2. The order pattern “**precedes**” specifies that the first event (e.g. Broker.order) is a necessary pre-condition which enables the occurrence of the second event (e.g. Buyer.orderConfirmation). The order pattern “**leads to**” describe cause-effect relationships; the first event – the cause (e.g. Buyer.receiveDelivery) must eventually be followed by the second event – the effect (e.g. Broker.receivePayment). Being in plain English, it is easier for business customers to understand and

cooperate with programmers to define IRS constraints from a set of business rules.

```

contract protocol BB
{
  Broker.order precedes Buyer.orderConfirmation globally;
  Buyer.orderConfirmation precedes Buyer.receiveDelivery
globally;
  Buyer.receiveDelivery leads to Broker.receivePayment
globally;
  Broker.receivePayment precedes Buyer.receivePaymentAck
globally;
}

```

Figure 4: Buyer-Broker protocols in Specification Pattern Systems

```

contract protocol BB
{
  Broker.order precedes Buyer.orderConfirmation globally
  where Broker.order.orderNumber =
    Buyer.orderConfirmation.orderNumber; //rule BB.1
  Buyer.orderConfirmation precedes Buyer.receiveDelivery
globally
  where Buyer.orderConfirmation.orderNumber =
    Buyer.receiveDelivery.orderNumber; //rule BB.2
  Buyer.receiveDelivery leads to Broker.receivePayment
globally
  where Buyer.receiveDelivery.orderNumber =
    Broker.receivePayment.orderNumber; //rule BB.3
  Broker.receivePayment precedes Buyer.receivePaymentAck
globally
  where Broker.receivePayment.orderNumber =
    Buyer.receivePaymentAck.orderNumber; //rule BB.4
}

```

Figure 5: Buyer-Broker protocol in Interaction Rule Specification

```

contract protocol BV
{
  Vendor.orderProcessing precedes Broker.orderConfirmation
globally
  where Vendor.orderProcessing.orderNumber =
    Broker.orderConfirmation.orderNumber; // rule BV.1
  Broker.orderConfirmation leads to Vendor.receivePayment
globally
  where Broker.orderConfirmation.orderNumber =
    Vendor.receivePayment.orderNumber; // rule BV.2
  Vendor.receivePayment precedes Broker.receivePaymentAck
globally
  where Vendor.receivePayment.orderNumber =
    Broker.receivePaymentAck.orderNumber; // rule BV.3
  Broker.receivePaymentAck leads to Vendor.sendOutBooks
globally
  where Broker.receivePaymentAck.orderNumber =
    Vendor.sendOutBooks.orderNumber; // rule BV.4
}

```

Figure 6: Broker-Vendor protocols in Interaction Rule Specification

However, the SPS does not consider the parameters of the operations/events. It is argued that the effect of different parameter values should be taken into consideration with the service interaction logic [2, 8]. The Buyer should be able to place multiple orders while waiting for deliveries of previous orders.

Therefore, the IRS [8] addresses this limitation of SPS by adding the **where**-clause associating the specific operation invocations of interest and relationship between their parameters/return values. Figure 5 and Figure 6 show the complete specifications of the protocols in Buyer-Broker (BB) and Broker-Vendor (BV) contracts. The **where**-clauses in the specifications mean that each book order request will trigger the creation of a different context keeping track of the protocols, and the protocols have to be followed within that given order.

The dependencies between protocols in contracts in the composition are also important to ensure that the interactions are carried out correctly. For example, in the case that the Broker sends order confirmation to Buyer before it receives the order confirmation from Vendor; the Buyer-Broker and Broker-Vendor protocols are still correct, but the correctness of the transaction as a whole has been violated. Figure 7 shows an example of dependencies between Buyer-Broker (BB) contract and Broker-Vendor (BV) contract.

```

protocol dependencies BB_BV
{
  BB.Broker.order precedes BV.Vendor.orderProcessing globally
  where BB.Broker.order.orderNumber =
    BV.Vendor.orderProcessing.orderNumber; // rule BB_BV.1
  BV.Broker.orderConfirmation leads to
  BB.Buyer.orderConfirmation globally
  where BV.Broker.orderConfirmation.orderNumber =
    BB.Buyer.orderConfirmation.orderNumber; // rule BB_BV.2
}

```

Figure 7: Dependencies between BB contract and BV contract

5. Protocol aggregation

A service is to play a certain role in the composition. In order to verify the compatibility between service's protocol and the required protocol, a role-centric aggregation of contract protocols and their dependencies is required. In order to facilitate the dynamic changes of protocols, the role-centric protocol aggregation is done automatically by a parser generated by the ANTLR Parser Generator [16], then the consistency of the role-centric protocol will be checked to ensure the correctness of the aggregation. This is performed dynamically by the organiser.

We define the structure of Interaction Rule Specifications with a grammar (lexer and parser in ANTLR terminology). The ANTLR Parser Generator then generates the code of a parser based on that grammatical structure description. The parser interprets the protocol descriptions in the contracts associated with a given role and their dependencies managed by the organiser, and then it produces a description of the role-centric protocol for that role. As an illustration, Figure 8 details the required protocol for the Broker role which is an aggregation of the protocols in Figure 5, Figure 6 and Figure 7. For simplicity, the **where**-clauses have been omitted from the protocol specification. Then these constraints are checked for consistency by the organiser.

```

role protocol Broker
{
  Broker.order precedes Buyer.orderConfirmation globally;
  // from BB.1
  Broker.order precedes Vendor.orderProcessing globally;
  // from BB_BV.1
  Broker.orderConfirmation leads to Buyer.orderConfirmation
  globally;
  // from BB_BV.2
  Vendor.orderProcessing precedes Broker.orderConfirmation
  globally;
  // from BV.1
  Broker.orderConfirmation leads to Vendor.receivePayment
  globally;
  // from BV.2
  Vendor.receivePayment precedes Broker.receivePaymentAck
  globally;
  // from BV.3
  Broker.receivePaymentAck leads to Vendor.sendOutBooks
  globally;
  // from BV.4
  Buyer.receiveDelivery leads to Broker.receivePayment
  globally;
  // from BB.3
  Broker.receivePayment precedes Buyer.receivePaymentAck
  globally;
  // from BB.4
}

```

Figure 8: Broker role-centric protocol aggregation (where-clauses omitted)

In order to perform the consistency check, it is required to define formal semantics for the temporal constraints. There are various formal mappings as listed in [15], such as Linear Temporal Logic, Computation Tree Logic, Quantified Regular Expression, etc. However these formalisms require a strong expertise level and mathematical background which many if not most developers are not familiar with. We adopt Finite State Automata (FSA) based semantic model for the IRS as defined in [8] to formally express the constraints specified in contracts, the dependencies between contracts and the aggregated role-centric views in roles. Each constraint pattern has a corresponding FSA as defined in [8]. The FSAs and their composition will form the basis for consistency and compatibility reasoning.

The FSA translations and consistency check are done by the FSA Generator in the organiser. Figure 9 illustrates this process.

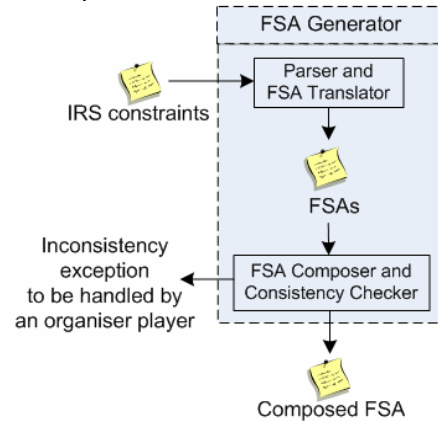


Figure 9: FSA Generator in an organiser

The protocol specification in Figure 8 is input into the Parser and FSA Translator (PFT) which parses the constraints and produces an FSA corresponding to each constraint. Then the consistency check is carried out by the FSA Composer and Consistency Checker (FCCC) based on the generated FSAs. The PFT and the FCCC are part of the FSA Generator. The FCCC will compose the component FSAs and compute the FSA intersection. An empty intersection implies the existence of conflicting constraints [17]. If such conflict exists, an inconsistency exception will be thrown and handled by the organiser player. The level of intelligence of specific organiser players can vary. It can throw the exception again which will be handled at a higher level of the composition. Alternatively, it can try to apply different protocols combination based on a knowledge set as in the case of an intelligent agent. If it failed to do so, a human intervention is required. How an organiser player specifically deals with the exception is out of scope of this paper.

6. Compatibility checking and run-time monitoring

For a service to play a particular role, the service's protocol has to be compatible with the role-centric protocol. If none of the available services satisfy the required protocol, the organiser may search for other services or a negotiation and redesign process can be carried out.

Protocol compatibility checking is performed when a service is bound to a given role by the organiser. Services interact with each other via role interactions. To enforce the predefined protocols, role interactions are intercepted and checked for

conformance. Figure 10 shows the overall architecture of the compatibility checking and run-time monitoring mechanism.

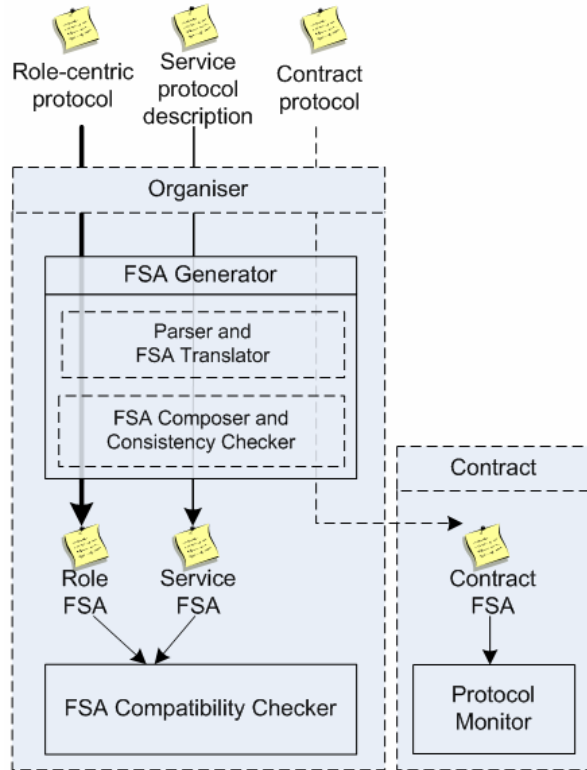


Figure 10: Compatibility checking and run-time monitor tool

During the compatibility check, the organiser retrieves both the protocol description of the service (e.g. in OWL-S description), and the role-centric protocol. These descriptions are then passed to the FSA Generator tool in the organiser that translates and generates composed FSAs corresponding to those specifications. The two FSAs produced by the FSA Generator are then input into the FSA Compatibility Checker (FCC). Within an FSA, states are nodes and events are edges connecting between nodes. We distinguish between input and output events. The two entities are said to *be able to communicate* when their FSAs has common events (the intersection of their events is not empty) and the composed FSA has traces to final state(s). *Trap state* occurs when there exists an output event produced by one component at a certain state is not consumable by the other FSA (i.e. the input event set of the other FSA at a specific state does not contain that output event) [8]. The two entities are *compatible* when they are able to communicate and their composed FSA does not have any traces of events which lead to trap states.

The **run-time monitoring** is performed by the Protocol Monitor (PM). The PM is built upon the contract interceptor as described in Section 3. The PM has a composed FSA that expresses the contract's constraints. When an interaction is intercepted, the PM will check the parameters according to the specification in the **where**-clauses. For each different value of the parameter, it will generate a new coordination context containing an instance of the composed FSA that tracks the conversation. On the other hand, if the parameter value is already encountered, it will update the existing coordination context's FSA. The interaction is allowed if it is consistent with the states in the FSA in a specific coordination context. Otherwise the interaction will be rejected and the composite's organiser is informed of the contract violation.

7. Flexibility of changing protocols

Let us consider the case that the library wants to change its policy so that it pays an order before receiving the delivery of that order. As our approach supports incremental specification, the modification can be performed at run-time by removing unwanted constraints and adding new constraints. In the Buyer-Broker contract protocol given in Figure 5, we remove the constraint BB.3 and add a new constraint to obtain the following (the **where**-clauses are omitted for simplicity):

```
contract protocol BB
{
  Broker.order precedes Buyer.orderConfirmation globally;
  Buyer.orderConfirmation precedes Buyer.receiveDelivery
globally;
  // Buyer.receiveDelivery leads to Broker.receivePayment
globally; // deleted
  Broker.receivePayment precedes Buyer.receivePaymentAck
globally;
  Buyer.receivePaymentAck precedes Buyer.receiveDelivery
globally; // newly added rule
}
```

The Broker-Vendor contract protocol remains unchanged. After changing the Buyer-Broker contract protocol, the automated aggregation mechanism will check for consistency and create new role-centric views of the protocols in the Buyer and Broker roles.

As dynamic modification of the system can affect any on-going transactions, safe points need to be defined so that the modification requests can only be carried out at these points in order to maintain system's consistency. When a modification request is received,

any incoming transactions will be queued. When all the outstanding transactions reach their final states, the modification request can then be carried out.

8. Related work

The need to add precise specification of protocol into interfaces of software components and Web services has been an active research area in recent years. In this section, we discuss the approaches to composition and coordination in SOC and in component-based software engineering.

In the Web service context, a number of standards have been developed that aim to provide composition and coordination of loosely coupled services. BPEL [18] has become a *de facto* standard for Web service orchestration. However, as pointed out in the introduction to this paper, BPEL's imperative programming paradigm means that protocols and their dependencies are hard-coded into the orchestration script. This script cannot cope with arbitrary changes in protocol. In general, BPEL tangles the definition of process with the management of that process (i.e. the orchestration code that defines the process is mixed with code for error handling, performance measurement, and any code for adaptation). IRS and BPEL are of different paradigms. IRS declaratively defines constraints on the occurrence and order of events; whereas BPEL takes an imperative approach by using sequencing constructs (such as **switch**, **while**, etc.) to specify a business process.

Standards that address Web service coordination have also gone some way in addressing protocols for both short-term 'atomic' and longer term 'business activity' transactions. For example, WS-Coordination [19] allows 'coordination contexts' to be created in which services can participate. These contexts provide a management framework for specific coordination protocols defined using standards such as WS-Transaction [20]. These standards provided 'external' coordination mechanisms to which services in different organisational domains can subscribe. In this paper, on the other hand, the focus is on contract protocols from the perspective of the compositional entity rather than the services themselves, and on the aggregation and generation of those protocols. In this way the ROAD approach is complementary to external mechanisms like WS-Coordination, because the role-centric protocol descriptions in ROAD still need to be mapped to the coordination mechanisms in 'outside world'. IRS constraints are 'internal' to a ROAD composition; hence, IRS specification does not introduce potential interoperability issues with services specification standards. When a service is bound to a role, the

organiser generates the mapping from service's protocol description to FSA and performs the compatibility check. The required mapping will be specific to each specification. There are some work in providing mapping from BPEL to Petri-net [21], and from BPEL to Deterministic FSA [22], we are currently working on the applicability of those approaches.

The SOAP Services Description Language (SSDL) is an extensible XML-based language for specifying service contracts underpinned by the Conditional Message Flows model [23, 24]. SSDL focuses on specifying messages and protocols to describe a SOAP-based Web Service; whereas our approach aims at defining the required protocols internal to a ROAD composition and different software components are dynamically bound to specific roles. Our framework supports and governs the communication among heterogeneous components while SSDL is specific to Web service. In addition, Conditional Message Flows (CMFs) define the message behaviour (protocol) by using Boolean conditions to specify when a message can be sent and received. There are certain advantages in using CMF compared to existing process modelling languages such as BPEL in terms of expressiveness. However, CMF does not support incremental specification of the protocols; hence the protocols have to be redefined if there are changes required at runtime.

WS-CDL (Choreography Description Language) [25] defines "behavioural interface" descriptions for services which are similar to role-centric protocols in ROAD. These descriptions are combined into a global view of the potential message exchanges between two or more services. These descriptions are then used to create code skeletons or BPEL abstract processes. As such, WS-CDL does not define a runtime infrastructure as does ROAD, and does not address runtime changes to protocols.

In the component-based software development area, Canal et al. [26] present a method of using a sugared subset of π -calculus to describe object service protocols and automated checking of protocol interoperability between CORBA objects. Although the π -calculus is expressive and powerful in describing the dynamic behaviour of objects and there are a lot of tool supports, it is difficult to define all possible interactions that are allowed to happen between two interactions of interest. Cho et al. [27] use UML Object Constraint Language (OCL) [28] to specify the pre and post-conditions for each method in the interface specification and use a state machine construct to describe protocols. The interoperability testing framework proposed a useful way to

automatically generate test cases to verify the component interaction. However, in their approach, there are strong dependencies between interface specification and the protocol specification, which limits the modification of protocol and reuse of interface specification. Reussner [29] also presents a model of software component interfaces using an extension of finite state machines. His approach is aimed at facilitating the dynamic adaptation of components during composition. From the linking of provides-interface and requires-interface, a component can adapt its published list of offered functionality in case not all the required external components are available. Similar to our approach, [30] uses temporal operators to specify the interaction constraints of a component relative to other components. Our approach differs from [30] in its ability to specify the dependencies between protocols and to produce consistent aggregated views of protocols. In general, our approach is complementary to the above approaches in that a component can be bound to a role, thus our approach can also provide an interoperation framework to integrate software components.

9. Conclusion and Future work

When composing services, it is important to ensure that the component services are matched in terms of not only the interface signature of the operations, but also the protocol specifications of the services. In a service composition, in addition to the point-to-point protocols, the dependencies between multi-party communications are also important. In this paper, we have presented a method to specify the protocols between point-to-point interaction and their dependencies by using Interaction Rule Specification (IRS), an extension of the Specification Pattern System (SPS). Those protocol specifications are then aggregated dynamically by an automated mechanism to produce consistent views of the protocol. This mechanism is provided by the ROAD (Role Oriented Adaptive Design) framework in order to provide the adaptability and flexibility to changing protocols. The extension to ROAD described here also provides a run-time monitoring mechanism so that the conformance to the predefined protocols can be verified dynamically.

The mechanism to translate IRS constraints to FSA and perform consistency checks based on the generated FSAs (ref. Section 5) has been implemented. The compatibility checking between service's and role's protocols and run-time monitoring (ref. Section 6) will be completed in the near future. In our prototype, we tried to compose the FSAs of the Broker

role-centric protocol as specified in Figure 8; however the run-time overhead is quite substantial and optimisations will need to be done.

IRS is expressive in specifying ordering and occurrence of business events, however, for a business transaction to be viable, time constraints must also be considered [31]. For example, the payment has to be made within 10 days of delivery. Adding time constraints into IRS is one of our priorities.

While the approach described here shows how protocols are defined, checked for compatibility and monitored, further work needs to be done on defining strategies for resolving mismatches between protocols, and creating automatic mechanisms for implementing those strategies. In order to achieve a higher degree of adaptability of protocols, we are investigating two broad approaches. The first is the creation of adaptor generation mechanisms between the roles and the services so that the mismatches in the services' protocols and the required aggregated protocols can be resolved. The second approach is to make changes to the role structure, for example to create additional mediator services that can resolve protocol mismatches. This approach would require an intelligent agent which can reason about and have access to a collection of composition structures and protocol specifications. We will explore these challenges in the future.

10. References

- [1] M. P. Singh and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, Chichester: John Wiley & Sons Ltd., 2005.
- [2] Z. Li, J. Han, and Y. Jin, "Pattern-based specification and validation of Web services interaction properties," in *Proc. of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*, Amsterdam, The Netherlands, 2005, pp. 73–86.
- [3] J. Han, "Temporal logic based specification of component interaction protocols," in *New Issues of Object Interoperability*, Caceres, Spain, 2000, pp. 43–52.
- [4] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins, "Making components contract aware," *IEEE Computer*, vol. 32, no. 7, pp. 38–45, 1999.
- [5] P. Collet, "Functional and non-functional contracts support for component oriented programming," in *First OOPSLA Workshop on Language Mechanisms for Programming Software Components, OOPSLA 2001*, Tampa Bay, Florida, US, 2001.
- [6] A. Colman and J. Han, "Using role-based coordination to achieve software adaptability," *Science of Computer Programming*, vol. 64, no. 2, pp. 223–245, 2007.
- [7] A. Colman, L. D. Pham, J. Han, and J.-G. Schneider, "Adaptive application specific middleware," in

MW4SOC Workshop at the 7th International Middleware Conference, Melbourne, Australia, 2006.

- [8] Y. Jin and J. Han, "Consistency and interoperability checking for component interaction rules," in *Proc. of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Taipei, Taiwan, 2005, pp. 595–602.
- [9] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proc. of the 21st International Conference on Software Engineering (ICSE)*, Los Angeles, California, United States, 1999, pp. 411–420.
- [10] S. Becker, S. Overhage, and R. Reussner, "Classifying software component interoperability errors to support component adaptation," in *Proc. of Component-Based Software Engineering: 7th International Symposium (CBSE 2004)*, Edinburgh, UK, 2004, pp. 68–83.
- [11] D. Garlan, R. Alan, and J. Ockerbloom, "Architectural mismatch, or, why it's hard to build systems out of existing parts," in *Proc. of the 17th International Conference on Software Engineering (ICSE)*, Seattle, Washington, 1995, pp. 179–185.
- [12] M. Shaw, "Architectural issues in software reuse: it's not just the functionality, it's the packaging," in *Proc. of the 1995 Symposium on Software Reusability*, Seattle, Washington, United States, 1995, pp. 3–6.
- [13] A. Colman and J. Han, "Using association aspects to implement organisational contracts," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 150, no. 3, pp. 37–53, 2006.
- [14] K. Sakurai, H. Masuhara, N. Ubayashi, S. Matsuura, and S. Komiya, "Design and implementation of an aspect instantiation mechanism," *LNCS Transaction on Aspect-Oriented Software Development*, vol. 3880, pp. 259–292, 2006.
- [15] H. Alavi, G. Avrunin, J. Corbett, L. Dillon, M. Dwyer, and C. Pasareanu, The Patterns. [Online]. Available: <http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml>
- [16] T. Parr, Aother Tool for Language Recognition (ANTLR). [Online]. Available: <http://www.antlr.org/>
- [17] Y. Jin and J. Han, "Specifying interaction constraints of software components for better understandability and interoperability," in *Proc. of the 4th International Conference on COTS-Based Software Systems (ICCBSS'05)*, Spain, 2005, pp. 54–64.
- [18] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, Business Process Execution Language for Web Services version 1.1. [Online]. Available: <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>
- [19] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey, Web

Services Coordination (WS-Coordination) version 1.0. [Online]. Available:

- <ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>
- [20] Web services transactions specifications. [Online]. Available: <http://www-128.ibm.com/developerworks/library/specification/ws-tx/>
- [21] H. M. W. Verbeek and W. M. P. van-der-Aalst, "Analyzing BPEL processes using Petri nets," in *Proc. of the 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB 2005)*, Miami, Florida, USA, 2005, pp. 59–78.
- [22] A. Wombacher, P. Fankhauser, and E. Neuhold, "Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery," in *Proc. of the IEEE International Conference on Web Services (ICWS'04)*, San Diego, California, USA, 2004, pp. 316–323.
- [23] D. Kuo, A. Fekete, and P. Greenfield, "Expressing and reasoning about service contracts in service-oriented computing," in *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, Chicago, IL, USA, 2006, pp. 915–918.
- [24] SSDL - The SOAP Service Description Language. [Online]. Available: <http://ssdl.org/>
- [25] N. Kavantzaz, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, Web Services Choreography Description Language Version 1.0. [Online]. Available: <http://www.w3.org/TR/ws-cdl-10/>
- [26] C. Canal, L. Fuentes, E. Pimentel, J. M. Troya, and A. Vallecillo, "Extending CORBA interfaces with protocols," *The Computer Journal*, vol. 44, no. 5, pp. 448–462, 2001.
- [27] I.-H. Cho and J. D. McGregor, "Component specification and testing interoperation of components," in *Proc. of IASTED 3rd International Conference on Software Engineering and Applications*, 1999, pp. 27–31.
- [28] J. B. Warmer and A. G. Kleppe, *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.
- [29] R. Reussner, "Enhanced component interfaces to support dynamic adaptation and extension," in *Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, 2001, pp. 9043–9053.
- [30] J. Han and K. K. Ker, "Ensuring compatible interactions within component-based software systems," in *Proc. of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference (APSEC)*, Chiang Mai, Thailand, 2003, pp. 436–446.
- [31] B. Benatallah, F. Casasti, F. Toumani, and R. Hamadi, "Conceptual modeling of Web service conversations," in *Proc. of 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Berlin, Germany, 2003, pp. 449–467.