A UML2 Profile for Service Modeling

Vina Ermagan and Ingolf H. Krüger

[1] University of California San Diego
9500 Gilman Drive, Mail Code 0404, La Jolla, CA 92093-0404, USA
{vermagan, ikrueger}@ucsd.edu
http://sosa.ucsd.edu

Abstract. In this article we provide an embedding of an interaction-based service notion into UML2. Such an embedding is needed, because to this date, UML2 has only limited support for services – they are certainly not first-class modeling elements of the notation. This is despite the ever increasing importance of services as an integration paradigm for ultra large scale systems. The embedding we provide rests on two observations: (i) services are fundamentally defined by component collaborations; (ii) to support a seamless development process, the service notion must span both logical and deployment architecture. To satisfy (i) and (ii) we introduce modifications to the UML that focus on interaction modeling, and the mapping from logical to deployment service architectures. The result is a novel and comprehensive UML2 profile for service-oriented systems.

Keywords: Rich Services, Service-oriented Architectures, Web Services, Model Driven Architectures.

1 Introduction

A major challenge in the development of ultra large scale software intensive systems is the controlled integration of multiple subsystems, such that the resulting system fulfills a wide spectrum of integration requirements ranging from authentication to security to policy management and governance. *Web services* have proven useful as a lightweight deployment and implementation mechanism for system integration; support for many of these integration challenges is, however, still under development in the Web services community. Furthermore, little guidance exists to date on how to model and design service-oriented architectures such that they leverage the emerging standards, such as WS-Security (authentication and security) and WS-BPEL (business process modeling and execution), as part of an integration solution. However, service-orientation is quickly gaining ground also in other domains with increasing software complexity; the automotive domain is one example, where service-orientation is a declared goal [21] but the deployment architectures are quite removed from a Web services flavor.

Contributions: This paper addresses this challenge by introducing a UML2 profile for the specification of service-oriented architectures that can be deployed on a variety of different object-, component- and service-oriented platforms. In particular,

G. Engels et al. (Eds.): MoDELS 2007, LNCS 4735, pp. 360–374, 2007.

to mention two extremes, service-oriented models according to our profile can be directly mapped not only into a Web service-enabled environment, but also into purely component-oriented deployment environments such as in automotive or avionics.

To that end, we develop a modest set of stereotypes with associated structural and behavioral rules. To address the integration challenges of the system class we target, we place the interplay of the constituent services in the center of concern. Therefore, a major means for specifying services in our approach is by means of interaction diagrams. However, we allow the full set of structural and behavior specification techniques to describe service interfaces and detailed service behaviors.

Figure 3 shows a generic example of the decomposition of a service-oriented architecture according to the *Rich Services Profile* we define in this paper. Intuitively, the profile introduces services as having an interface to their environment and, if they are composite, a predefined internal structure. This internal structure is modeled after two major successful architectural patterns: (1) the emerging Enterprise Service Bus (ESB) and Message-Oriented Middleware (MOM) technologies, such as the increasingly popular Mule/ActiveMQ [22] combination; (2) bus-oriented industrial communication architectures, such as they are found in production plants, cars and airplanes. The basic idea is that every service consists internally of a messaging component, a router, and a set of internal services. Any call upon the service (which we model as a message sent to the service via its Service/Data adapter) is intercepted by the Router, which – using the Messenger as its communication infrastructure – exposes the message to a prescribed set of internal services. Each such internal service can alter or transform the message on its path to its final destination. Analogously, calls made by the Rich Service are also intercepted by the Router before they leave via the Service/Data Adapter. This architectural blueprint provides a rich control framework for service composition.

Benefits: The immediate benefits of this profile are as follows: (i) The concept of service is introduced as a first-class modeling citizen into the UML – in particular, service interfaces and service behavior can be modeled using the well-known description techniques provided by the UML. (ii) The profile defines a structural and behavioral blueprint for controlled service composition and refinement – each composite service can define a set of interaction protocols that govern the interplay of its constituent services, so that the interplay addresses the functional and non-functional integration requirements. Each service, in turn, can be hierarchically decomposed according to the same blueprint to support scalability of the modeling approach. (iii) The distance between a logical service-oriented architecture following the blueprint and a suitable deployment architecture is minimal, resulting in improved traceability from requirements to implementation.

Outline: The remainder of this paper is structured as follows. In Section 2, we introduce the Rich Service Profile in detail by introducing a structural and behavioral domain model for Rich Services modeled after Figure 3. In particular, we describe the stereotypes we introduce, their interplay in terms of behavioral constraints, and our rationale for selecting the design decisions we made. Along the way we also mention the description techniques available to the engineer in specifying systems according to the profile. Section 3 presents a case study illustrating both the modeling approach

enabled by the profile, and the use of the stereotypes; the context of the case study is a large-scale system-of-systems integration architecture in the domain of ocean observatories. In Section 4, we discuss our approach in the context of related work. Section 5 contains our conclusions and outlook.

2 Rich Service Profile

As service-oriented modeling and implementation technologies become more popular, so does the need for systematically designing large-scale systems of systems integration solutions based on services. The UML is a common and widely used set of notations providing visual modeling languages, valuable for modeling, design and comprehension of requirements and architectural designs. Currently, the UML supports specific notations for development of object- and component-oriented software, but to date, no explicit notion of service, as a first class modeling entity, is defined in the UML.

The profile mechanism has been specifically defined for providing a lightweight extension mechanism to the UML standard for tailoring UML for various domains or different target platforms. Stereotypes, tagged values, and constraints are the main extension mechanisms available in a profile. To complete the previous versions of UML, the UML2 infrastructure and superstructure specifications have defined the profile mechanism as a specific meta-modeling technique, where stereotypes are specific metaclasses, tagged values are metaattributes, and profiles are specific packages [23]. In this section, we take advantage of the UML2 profile package to create a profile as a metamodel for complex service-oriented architectures.

The goal of the Rich Service Profile we propose here is to provide a common language for describing the central aspects of service-oriented systems. This includes specification of the syntactic *and* semantic interface of individual services, behavior specifications for services, service composition, and the mapping of services to deployment architectures. As mentioned in Section 1, our particular focus is the controlled aggregation of individual services into composite service architectures, such that the resulting architecture, by construction, observes a wide spectrum of crosscutting requirements. The profile we present supports a variety of deployment platforms for implementation of the modeled service(s) – including traditional web service-based approaches, emerging Enterprise Service Bus technologies, and general message-oriented middlewares.

In this section, we utilize the standard mechanism for tailoring the UML, *profiles*, to provide the core of a common language supporting the mentioned goals.

The Rich Service Profile references the UML metamodel as its reference model. It extends Components to specify *Rich Services* and further constructs, including Router, Messenger, and Service Interfaces, needed for supporting them. In essence, a Rich Service serves as a Wrapper around traditional services, including web services, within an architectural framework that supports hierarchical service decomposition, as well as addressing composition and integration concerns within and across hierarchical levels.

The profile also includes collaborations that define the general behavior of the main entities of the profile. These collaborations serve as guidelines for designers

who can further refine the general behaviors present in the profile to create a more detailed deployment model.

In the following subsections, we introduce the stereotypes of the Rich Service Profile together with the relevant collaborations in detail.

2.1 Rich Service Profile Stereotypes

The stereotypes of the Rich Service Profile and their base classes are described in Table 1. Figure 1 illustrates the metamodel that the Rich Service Profile provides using the stereotypes of Table 1.

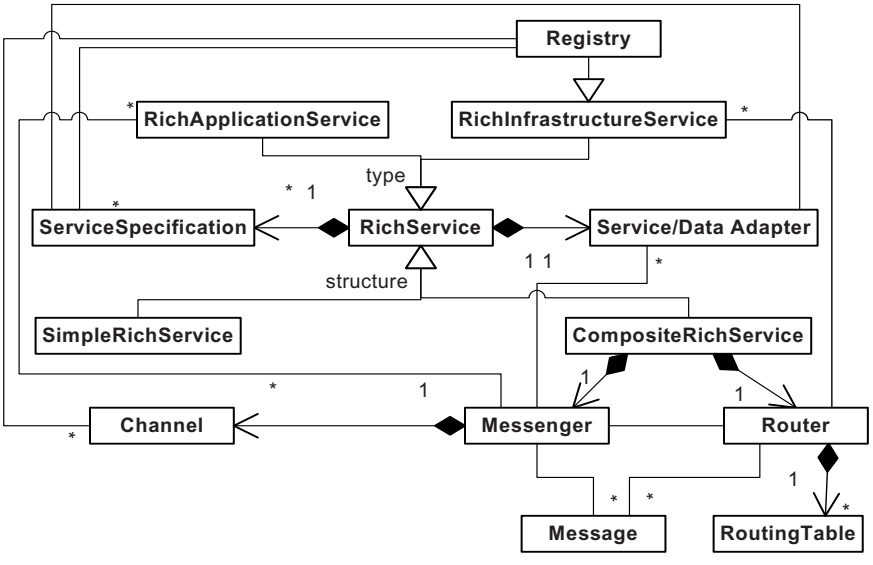

Fig. 1. The Rich Services metamodel

The central entity of the profile is the *Rich Service*. It serves to model individual services, as well as their integration into composite services. Intuitively, a Rich Service consists of the following entities: (1) a *Service/Data Adapter*, which serves as the interface of the Rich Service to its environment, (2) a *Messenger*, which is responsible for message transmission among the sub-services of a Rich Service, (3) a *Router*, which is responsible for intercepting inbound and outbound messages to and from the Service/Data Adapter and routing these messages through the correct set of sub-services, and (4) the *sub-services* themselves, which are also Rich Services that communicate using the Messenger and Router.

A Rich Service is modeled as a stereotype extending Component from the UML BasicComponents package. A Rich Service is active, meaning that it has an associated behavior; it has precisely one externally visible port stereotyped as the Service/Data Adapter. A Rich Service defines provided and required Service Specifications. Service Specification stereotypes the Interface from the

ProtocolStateMachines package and has a tag named Protocol, which is a protocol state machine that defines the external view of the sequence of operation calls that can occur on the interface.

Table 1. Stereotypes of the Rich Service Profile

Stereotype	Base Class	Tags	Parent
RichService	Component	Adapter	
SimpleRichService	Component		RichService
CompositeRichService	Component	Messenger, Router	RichService
Messenger	Component		
Router	Component	RoutingTable	
RoutingTable	Class		
Channel	Class		
PublishSubscribe Channel	Class		Channel
PointToPointChannel	Class		Channel
DataTypeChannel	Class		Channel
Message	NamedElement		
Adapter	Port (From ProtocolStateMachines)	Protocol	
ServiceSpecification	Interface (from ProtocolStateMachines)	Protocol	
Registry	Component	Publish: (Provided Interface)	

A Rich Service can be simple, meaning that it has no (or, more precisely, a trivial) internal structure in the sense of entities (2)-(4) mentioned above; it can also be composite, having an internal structure as follows. A Composite Rich Service has a Messenger, a Router, and a number of internal Rich Services. Messenger and Router are stereotypes extending Component. Multiple internal Rich Services can be attached to the Messenger via ports. Messenger is responsible for Message transmission between the connected Rich Services and between the Rich Services and the Service/Data Adapter. Messenger has a number of Channels to implement the messaging. A Channel is a stereotype extending Class. This allows the profile to support various types of channels, including Publish_Subscribe Channel, DataType Channel, and Point_to_point Channel. A Messenger is always associated with a Router. The Router is responsible for routing the messages through the correct set of Channels based on its Routing Tables. Routing Table is a stereotype extending Class. Intuitively, the router is the mechanism that allows us to inject monitoring and transformation services into a composite service. The idea is that the router intercepts inbound messages at the Service/Data adapter, before they are accessible to the internal Rich Services. The router then follows the configuration stored within its Routing Table to steer the processing of these messages from one internal Rich Service to another. This mechanism can be used, for instance, to encrypt or decrypt messages, to log them, to persist them, etc, without the sender being aware of the

intermediate services. Similarly, outbound messages are exposed to the routing scheme before they leave the Rich Service via the Service/Data Adapter.

Rich Services can be of two types: Rich Infrastructure Service (RIS), or Rich Application Service (RAS). Rich Application Services are only aware of the Messenger, while Rich Infrastructure Services can manipulate the Routing Tables and therefore have access to the Router. Rich Infrastructure Services are Rich Services that can directly access the routing tables in order to provide services to the messaging infrastructure, while Rich Application Services provide application-specific services to the system. A specific example of a Rich Infrastructure Service is a Registry where other Rich Services can publish their Service Specification, i.e. their interfaces including the protocol state machines. The Registry associates Channels to published Service Specifications; other Rich Services can subscribe to Channels based on their Service Specifications. The information on subscription of Rich Services to Channels is kept as part of the Routing Table and the Router is responsible for routing the messages sent by the provider Rich Service to the subscribing Rich Services. When a Service/Data Adapter puts a Message on the Messenger, the Router intercepts the Message and routes it based on the Routing Table information through a set of Rich Infrastructure and Application Services.

2.2 Behavior

Collaborations are particularly useful as a means for capturing standard design patterns. Since a Collaboration in UML2 is a kind of classifier, any kind of behavioral description can be attached to it. By extending Collaborations from the UML2 Collaborations package we can form prototypical collaborations to define behavioral pattern of some of the Rich Service Profile entities as part of the profile. These Collaborations can have associated interactions to achieve a more detailed behavior specification. The Stereotyped Collaborations can be used as guidelines for designers on how to use and integrate the profile entities to form meaningful system models.

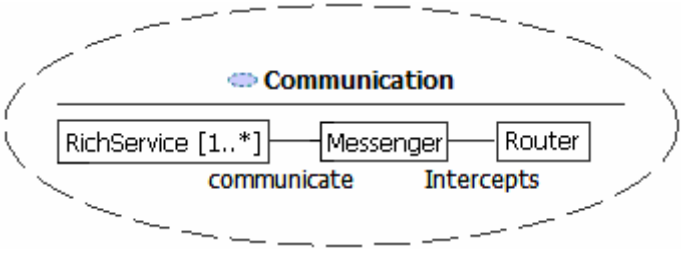

Fig. 2. Communication collaboration

A Communication Collaboration for a Composite Rich Service (see Figure 2) is a stereotype that has a Messenger, a Router, and multiple Rich Services as its parts (tags). Every Composite Rich Service instantiates such a collaboration. The bindings of the collaboration roles to the Rich Service's parts are trivial due to the shared names of the roles and Rich Service's parts. An interaction can be attached to this collaboration, specifying the behavior of the Router as an interceptor (Smart Proxy

[29]). Every Rich Service can send a Message to Messenger. The Router works as an interceptor and picks up the Message, routs it through any specified intermediate Rich Services before sending it to the destination Rich Service. These intermediate Rich services can be Rich Infrastructure Services, or they can be other Rich Application Services. This describes the generic behavior for service composition. A composite Rich Service implements this behavior via the respective role bindings.

Designers can capture the overall behavior of a Composite Rich Service as an interaction. Such an interaction will have the internal Rich Services, the Messenger, and Router as its lifelines. This high level behavior will specify the order in which Rich Services communicate, and can be used to populate the Routing Table. To further refine the model behavior, one can use PartDecomposition from the UML2 Interactions package to decompose the internal Rich Services (modeled as lifelines) to capture the internal behavior of these internal Composite Rich Services. Of course, the internal behavior is visible from outside the Composite Rich Services only to the degree it is specified in the corresponding Service/Data Adapter. The Formal Gates on the decomposed interaction form the interfaces for the Composite Rich Service.

The high level behavior can also be represented as UML2 Protocol State Machines, which can be further redefined to form the internal behavior of encapsulated Composite and Simple Rich Services. This allows us to model service behavior with all the behavior description techniques provided by UML2.

3 Case Study

We demonstrate the utility of the proposed profile and metamodel by using a case study from the domain of global ocean observatories, namely the federated Ocean Research Interactive Observatory Networks (ORION) program [24]. This case study is an elaboration of the ORION-CI conceptual architecture available at [24]. Clearly, here we can only scratch the surface of the complexity of building an architecture of the scale of ORION. However, it allows us to show (i) modeling of services and their integration, (ii) service decomposition, and (iii) the direct deployment mapping from an instance of the profile to state-of-the art Web services technologies. Along the way we will also sketch the key steps of our iterative service elicitation and architecture definition process: (1) model use cases and their relationships, (2) identify the collaborations, interfaces, and associated integration constraints that define the services needed to support the use cases, (3) flesh out the service architecture using Composite and Simple Rich Infrastructure and Application Services as needed, following the integration requirements elicited in (2), (4) specify behaviors of Simple Rich Services as needed, or refine them into Composite Rich Services, (5) specify mapping from the entities in the Rich Service Profile to deployment entities to create an instance of the architecture. Iterate over (1)-(5) until the desired degree of detail is reached.

A system satisfying the goals of ORION would support scientific discovery by providing eligible oceanographers with ubiquitous access to instrument networks for sensing and actuation, computational resources, and modeling and simulation facilities, as well as means for distributed data storage and access. A traditional SOA approach would quickly reach its limits in the face of the challenges induced by the

diverse requirements of supporting governance of the different authority domains, access policies, and concerns of the multiple stakeholders involved in such a complex system-of-systems. To capture the requirements for and manage the complexity of the resulting cyber-infrastructure we exploit the Rich Service Profile as defined above; we directly benefit from its disentanglement of logical and deployment architectures for services because the various subsystems indeed rest on a wide spectrum of deployment technologies.

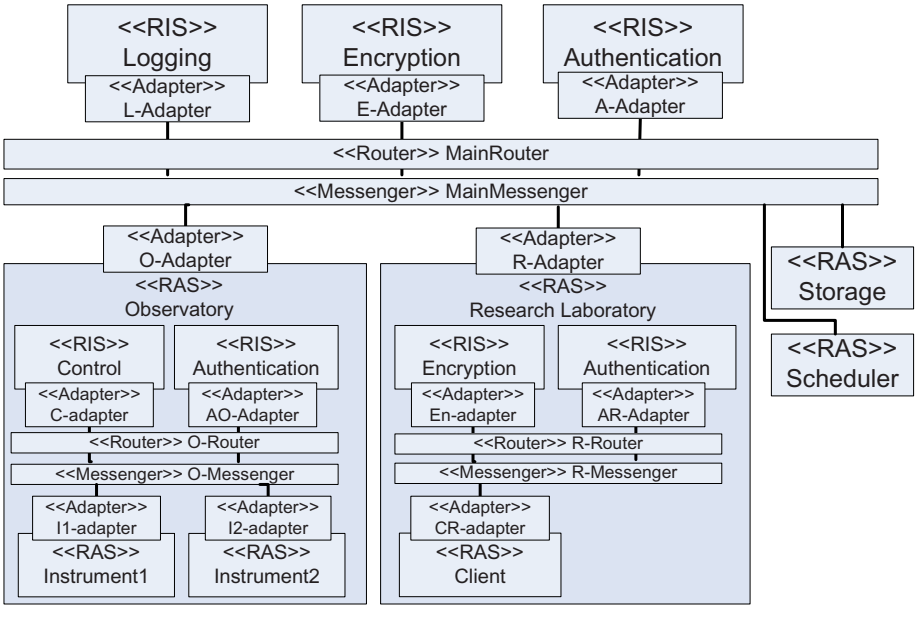

Fig. 3. Orion case study model based on our profile

The hierarchical nature of Rich Services supports creating traceable views for various stakeholders of the system, and a decomposition methodology that supports operation and distributed management of thousands of independently owned taskable resources (modeled as services) of various types (e.g., sensors, sensor platforms, processes, numerical models and simulations) across a core infrastructure operated by independent stakeholders. This also enables hierarchical structuring of the stakeholders' logical roles into the cyber-infrastructure, and encapsulation of crosscutting concerns according to their individual policies.

Figure 3 represents a possible subset of stakeholders as high-level Rich Services such as an *Observatory* and a *Research Laboratory*. Such a decomposition allows us to reason about their role in the cyber-infrastructure without dealing directly with their internal deployment models. Steps (1)-(3): In order to illustrate the steps involved in modeling such a system based on the proposed profile, we will consider *one* use case of the system, namely an oceanographer accessing a remote ocean instrument and retrieving the experimental data from the instrument. As a requirement for this use case, all of the conversations between an oceanographer and an instrument

must be logged. The oceanographer and instrument are parts of different authority domains, each with its own set of requirements and policies. At a very high level view, we can abstract from the Adapters and concentrate on the communication between Rich Services. The use case can be modeled as a Communicate collaboration (see Figure 2) use where Research Laboratory and the Observatory play the roles of Rich Services, and the Messenger and the Router play their respective roles.

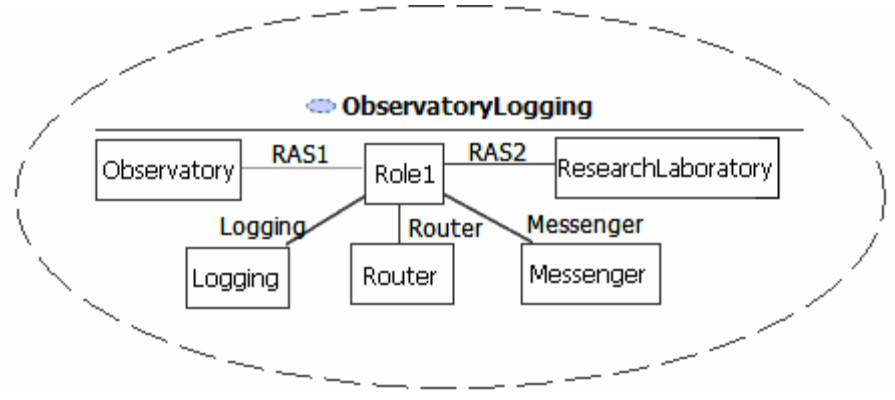

Fig. 4. Observatory-Research Laboratory Collaboration with Logging

To enforce the logging requirement, however, we create a logging collaboration, as it can be reused for other use cases of the system as well. The Logging collaboration has two Rich Application Service roles: RAS1 and RAS2, and a Logging Rich Infrastructure Service. It specifies that every message sent by RAS1 to the Messenger will be sent by the Router to the Logging role and then to the RAS2, through the same Messenger. An interaction diagram can be used to express the sequence of interactions for this collaboration in more detail. Now, we can create a new collaboration for our use case that uses the Logging collaboration to capture the communication of the Observatory and the Research Laboratory. This collaboration is shown in Figure 4. In order to capture the detailed behavior of this collaboration we use a UML interaction diagram shown in Figure 5.

In this interaction, the Research Laboratory, Observatory, Router, and Messenger are captured as lifelines, because they are connectable elements (parts) of their container, i.e. the Rich Service modeling the overall system. The Research Laboratory sends the request to the Messenger, destined for the Observatory. The Router intercepts this communication, sending the request to the Logging Rich Service, also via the Messenger. After being processed by the Logging service, the Router routes the request to the final destination, the Observatory. Note that by using UML interactions, we can further impose time and duration constraints on the occurrences of partial interactions. This interaction model captures the essential behavior of the system to fulfill the use case and its integration requirements and constraints.

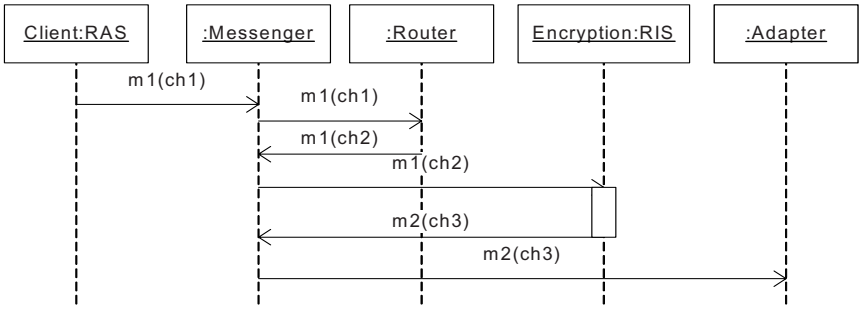

Fig. 5. Interaction diagram for the Observatory-Research Laboratory collaboration

Step (4) - behavior specification and refinement: The Research Laboratory itself is a composite Rich Service and may have many internal services such as oceanographer client service, identification, authentication, and encryption of the outbound messages. The identification and the management of the remote *Instrument* also concerns another stakeholder, as the *Instrument* is located deeper in the hierarchy of the *Observatory*, and within the local control domain of a *Regional Cabled Observatory* near the seashore. We can use PartDecomposition from the UML Interactions package, which is a new concept added in UML2, to further decompose the model capturing the interactions occurring between the internal parts (Rich Services) of each of these composite Rich Services. As an example, Figure 6 shows how the Oceanographer Client's outgoing requests should be intercepted by the internal Router and processed by the Encryption Rich Service before reaching the Adapter, which acts as a gateway for outbound messages.

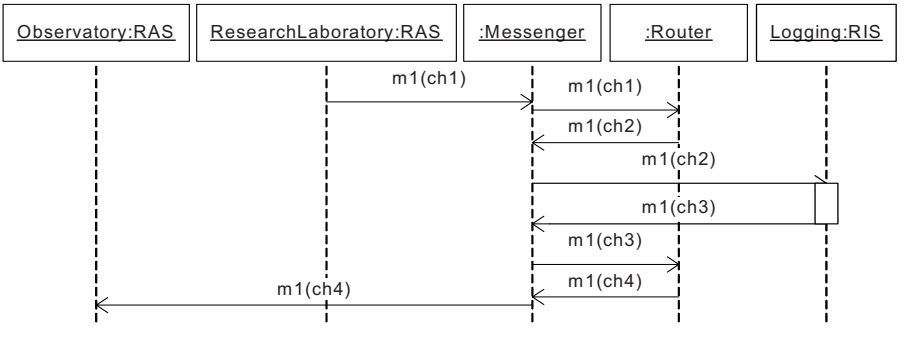

Fig. 6. Research Laboratory internal interaction for Encryption

Note that the only formal gates for the internal interactions of a composite Rich Service exist on the Adapter lifeline of the interaction. Following the semantics of the PartDecomposition, these formal gates must match the actual gates on the decomposed lifeline in the higher level interaction. Together these gates define the partial interfaces of the composite Rich Service. The union of the gates of the

composite Rich Service from all the interactions that it participates in will form the complete interface of this service. Since these gates are expressed as parts of the interactions, partial or global protocol state machines can be assigned to their union in the interface, giving a richer definition of the service.

Also note that PartDecomposition is used to model a form of service composition where the participating services are parts of the same composite Rich Service. Sanders et al. [14] propose a methodology for modeling and specifying service composition using UML2 collaborations. They also use interaction overviews, and state machines to specify the collaboration behavior in further details. Such an approach, being based on UML2, fits well with our Rich Service Profile and can be used to model service composition. Service composition can be addressed in a centralized way by adding a coordinating Rich Service that will orchestrate the participating services and with the help of the Router/Interceptor. BPEL4WS is the standard choice for such an approach if the Web Services is the target domain. Service composition can also be addressed in a distributed way, where choreography based languages such as WSCL or WS-CDL can be used when targeting the Web Services domain. Rich Services support these composition approaches while their encapsulation and hierarchy guide developers to focus on one hierarchical level at a time[28].

Step (5) – deployment mapping: A possible deployment plan for such a system might include classic Web services or a more general Enterprise Service Bus (ESB)-based technology. ESBs combine the strengths of message-oriented middleware; a flexible plugin architecture for processing messages to handle crosscutting concerns for a set of connecting Rich Services; and a rich set of data adapters/connectors to facilitate rapid connections between emerging and legacy data sources, applications and services. Examples of ESB implementations include Architect's Toolbox, Cape Clear's ESB, Fiorano ESB, Sonic ESB, SpiritSoft's Spiritwave, and CodeHaus' Mule. For instance, a web service based target platform, might consider WSDL as the Adaptor and Service Interface description. Also conversion rules described in [15] for converting UML models to WDSL can be used. Leveraging the many technologies supported by an ESB, including Mule as transport mechanisms, an Adaptor can publish itself via JMS, HTTP, SOAP, etc. Also, in this example we have abstracted from the Registry services, assuming that the binding is hard-coded into the Routing Tables, which are actually a *feature* of the ESB itself. Since the Registry is modeled as a Rich Service, the same modeling approach can be used for the interactions including a Registry, which can be mapped to a UDDI service as a target technology. This example shows how the Rich Service Profile allows specification of a service-oriented architecture at *both* the logical and the deployment level, using the description techniques already included in UML2. Because it disentangles logical and deployment aspects of services, the profile lends itself to the modeling of complex service architectures with heterogeneous deployment infrastructures.

4 Discussion and Related Work

Model driven design and development of systems is a well-established practice [19]. However, model driven approaches to service-oriented design of systems are still in

their early stages. Although UML [23] is a commonly used and widely accepted modeling language, it still has no explicit support for services and their auxiliary constructs. The work presented here leverages the experience we have gained in earlier research, where we have built a comprehensive Service Architecture Definition Language (Service-ADL) with services as first-class modeling citizens [25], and makes these concepts accessible within the UML. In particular, this allows us to also carry over the interaction modeling, behavior synthesis, and architecture exploration techniques we have built for Service-ADL into the UML context [1] [2].
Several other attempts exist to use existing UML constructs to model service-oriented applications. [3] uses UML class diagrams to design a general model for service-oriented architectures (SOAs) and uses collaboration diagrams and graph transformation rules for dynamic architecture reconfigurations. [4] proposes use of UML 2.0 collaboration diagrams for modeling web service collaboration protocols along with activity and interaction diagrams as more detailed modeling levels. Kim [26] investigates how UML diagrams can be used to graphically specify collaboration protocols with an automated mapping to BPSS. UMM [27] provides rich set of UML-based modeling concepts for business to business collaboration protocols and methodological guidance to move from requirements gathering to implementation design. All of these approaches suggest the use of UML modeling techniques for SOA modeling. Our work complements these efforts by embedding an explicit metamodel for SOAs into the UML.

Profiles, as the only lightweight means for extending UML, were leveraged in many approaches to address the lack of support for services. All of these profiles are based on UML 1.x, while the changes from previous UML versions to UML 2.0 have important benefit and impact on service modeling, which is leveraged in our Rich Service Profile. Electronic Services are proposed in [5] as services that are enriched with content and provision. A UML profile for Electronic Service Management Systems [5] is created as a framework for representing operational logic of e-services, providing a conceptual infrastructure for e-services development and management. However, ESMS does not address the business definition and engineering of services. The Enterprise Collaboration Architecture (ECA) defined as part of the UML profile for distributed object computing (EDOC) [6] provides a comprehensive framework for modeling of enterprise systems, while still no explicit notion for services exists in this profile. UML activity models are recommended for service composition modeling in this profile. A metamodel for WSDL is proposed in [8] along with a mapping to UML. Our Rich Service Profile, while based on UML 2.0, has explicit notions for services and adds an architectural pattern to service modeling, while it maps well to currently used Web services technologies, such as WSDL, UDDI and BPEL. For instance, we can use BPEL specifications not only for individual services, but also for the interaction model of composite Rich Services, and then derive the corresponding routing and interaction constraints from these specifications. Gardner [7] describes a UML profile for automated business processes and a mapping of the profile to BPEL4WS. He uses UML activity graphs for specifying the business processes. This profile can be used along with the Rich Service Profile if a mapping of the model to BPEL as a target technology is desired. The profile proposed in [20] is based on UML2. It models services as first class elements; however, as mentioned before, the Rich Services Profile goes beyond by adding a scalable architectural pattern enabling

the managed integration of multiple existing service composition and coordination approaches.

In the web services domain, there are several other approaches such as [9] suggesting the use of activities from UML to model web service composition, while [10] proposes a service composition model based on UML class diagrams. Thöne et al. [11] create a UML profile for web service composition and propose the UML_WSC language as a replacement for BPEL4WS. A different approach to service composition leverages rich ontologies that describe service characteristics. For example, the Semantic Web [12] community uses semantic annotations to reason about Web services by using languages such OWL-S [13]. In our Rich Service Profile, services are enriched by having hierarchies and following the specific architectural pattern as part of the metamodel. The use of a Router as an interceptor allows for modeling dynamic reconfiguration at runtime, if needed. Specifically, we can model a wide range of service composition operators, including sequencing, alternatives, repetitions, and parallelism, by means of the Routing Table of the composing Rich Service. More complex operators, such as the ones modeling interrupts, or service synchronization can be modeled using the Routing Table in conjunction with an additional Rich Infrastructure Service that monitors and manages the composition result. Service interfaces are already augmented by suggesting the addition of protocol state machines on them as part of the metamodel, and the profile has the capability of exposing richer interfaces and communication reconfiguration at run time, thereby enabling the use of ontology-based composition techniques.

Work on Web Services composition has highlighted its tight coupling with interaction modeling. [16] explores the use of Message Sequence Charts (MSC) to define interactions. [17], for example, presents a tool that transforms MSC to BPEL specifications to allow Web services composition. We leverage similar techniques to compose Rich Services. Toward this goal we have already experimented with the definition and composition of services based on MSC in [18].

5 Conclusions and Outlook

Service-oriented modeling and implementation are the centerpieces of modern system-of-systems integration approaches. Web services and related technology standards address many important issues of service *deployment*. The *modeling* of service-oriented integration architectures, independently from Web Services deployments, however, is still an area of active research and experimentation. To date there is no widely accepted modeling language for that purpose – specifically, the UML2 proper has not assigned "first-class modeling status" to the notion of service.

In this paper, we have identified the need for having an interaction-based, hierarchical service model that disentangles logical architecture from deployment concerns. We have introduced an UML2 profile for Rich Services. Rich Services introduce an explicit integration architecture, consisting of a messaging and routing component, which allows controlled composition of the internal sub-services that implement a service's behavior. This provides support for a wide spectrum of service composition operators and allows the designer to manage crosscutting aspects of an integration task – examples are: encryption, governance, and policy management.

Furthermore, the hierarchic decomposition of Rich Services allows us to scale service models to any desired level of detail. Using a systems-of-systems integration challenge from the domain of oceanography, we have demonstrated utility of the profile, as well as the direct mapping of Rich Service models to current Web Service-based technologies.

By construction, we leverage all of the UML's description techniques for system specifications based on our profile; tailoring these description techniques further to address dynamic architecture changes, as they are supported by our profile, is one interesting area of future work.

Acknowledgments. Our work was partially supported by the NSF within the project "ITR: Collaborative Research: Looking Ahead: Designing the Next Generation Cyber-infrastructure to Operate Interactive Ocean Observatories" (award OCE/GEO #0427924), as well as by funds from the California Institute for Telecommunications and Information Technology (Calit2). We are grateful to the anonymous reviewers for insightful comments.

References

1. Deubler, M., Krüger, I., Meisinger, M., Rittmann, S.: Modeling Crosscutting Services with UML Sequence Diagrams. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 522–536. Springer, Heidelberg (2005)
2. Ermagan, V., Krueger, I., Menarini, M.: Towards Model-Based Failure-Management for Automotive Software. In: Proceedings of the ICSE 2007 Workshop on Software Engineering for Automotive Systems (SEAS) (2007)
3. Baresi, L., Heckel, R., Thöne, S., Varró, D.: Modeling and validation of service-oriented architectures: application vs. style. In: Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003, ESEC/FSE, pp. 68–77 (2003)
4. Kramler, G., Kapsammer, E., Kappel, G., Retschitzegger, W.: Towards Using UML 2 for Modelling Web Service Collaboration Protocols. In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05) (2005)
5. Piccinelli, G., Emmerich, W., Williams, S., Stearns, M.: A Model-Driven Architecture for Electronic Service Management Systems. In: Proceeding of International Conference on Service Oriented Computing, pp. 241–255 (2003)
6. Enterprise Collaboration Architecture: (ECA) Specification. Version 1.0. formal/04-02-01 (February 2004), http://www.omg.org/docs/formal/04-02-01.pdf
7. Gardner, T.: UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, Springer, Heidelberg (2003)
8. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA Approach for Web Service Platform. In: Proceedings 8th International Enterprise Distributed Object Computing, pp. 58–70 (2004)
9. Skogan, D., Gronmo, R., Solheim, I.: Web Service Composition in UML. In: Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conference (EDOC), IEEE Computer Society Press, Los Alamitos (2004)

10. Orriëns, B., Yang, J., Papazoglou, M.: Model Driven Service Composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)
11. Thöne, S., Depke, R., Engels, G.: Process-Oriented, Flexible Composition of Web Services with UML. In: Proceedings of the International Conference on Conceptual Modeling (Workshops), pp. 390–401 (2002)
12. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 34–43 (2001)
13. OWL-S: Semantic Markup for Web Services (2004), http://www.w3.org/Submission/OWL-S/
14. Sanders, R., Castejón, H., Kraemer, F., Bræk, R.: Using UML 2.0 Collaborations for Compositional Service Specification. In: Proceedings of the 8th International Conference of Model Driven Engineering Languages and Systems, pp. 460–475 (2005)
15. Gronmo, R., Skogan, D., Solheim, I., Oldevik, J.: Model-driven Web services development. In: EEE'04, pp. 42–45. IEEE, Los Alamitos (2004)
16. Krüger, I.H.: Distributed System Design with Message Sequence Charts, Ph.D. dissertation, Technische Univer-sität München (2000)
17. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Tool support for model-based engineering of Web service compositions. In: ICWS 2005, pp. 95–102. IEEE, Los Alamitos (2005)
18. Broy, M., Krüger, I.H., Meisinger, M.: A Formal Model of Services. ACM Transactions on Software Engineering and Methodology (TOSEM) 16(1), 5 (2007)
19. Mellor, S., Clark, A., Futagami, T.: Special Issue on Model-Driven Development. IEEE Software 20(5) (2003)
20. IBM: UML 2.0 Profile for Software Services, http://www-128.ibm.com/developerworks/rational/library/05/419_soa/
21. Krüger, I.H., Nelson, E.C., Prasad, K.V.: Service-Based Software Development for Automotive Applications. In: Proceedings of the CONVERGENCE 2004. Convergence Transportation Electronics Association (2004)
22. http://mule.mulesource.org/wiki/display/MULE/Home
23. Object Management Group: UML 2.1.1 Superinfrastructure version 07-02-03, http://www.omg.org/cgi-bin/doc?formal/07-02-05
24. ORION Program Cyber Infrastructure, http://www.orionprogram.org/organization/committees/ciarch/
25. Ermagan, V., Huang, T.-J., Krüger, I., Meisinger, M., Menarini, M., Moorthy, P.: Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems. In: Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES'07), Informatik-Bericht 2007-01 (2007)
26. Kim, H.: Conceptual Modeling and Specification Generation for B2B Business Process based on ebXML. In: SIGMOD Record vol. 31
27. Hofreiter, B., Huemer, C., Naujok, D.: UN/CEFACT's Buisness Collaboration Framework- Motivation and Basic Concepts. In: Proceedings of the MKWI (2004)
28. Arrott, M., Demchak, B., Ermagan, V., Farcas, C., Farcas, E., Krüger, I.H., Menarini, M.: Rich Services: The Integration Piece of the SOA Puzzle. In: Proceedings of the IEEE International Conference on Web Services (ICWS), Salt Lake City, Utah, IEEE Computer Society Press, Los Alamitos (2007)
29. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Reading (2003)