A Planning-Based Approach for the Automated Configuration of the Enterprise Service Bus

Zhen Liu, Anand Ranganathan, and Anton Riabov

IBM T.J. Watson Research Center
{zhenl,arangana,riabov}@us.ibm.com

Abstract. The Enterprise Service Bus facilitates communication between service requesters and service providers. It supports the deployment of "message flows" from a service requester to one or more service providers. These message flows incorporate different functions such as routing, transformation, mediation, security and logging. In this paper, we propose an AI Planning-based approach for the automated construction of message flows between requesters and providers based on high-level goals specified by the enterprise architect or administrator. This automated construction of flows can be used either in the design phase where a developer or architect is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changed requirements. The planning model is based on tags, where goals, components, and links in the message flow are described using sets of tags. We describe the planning model and a case study that demonstrates the power of our approach in constructing flows in response to high-level requirements.

1 Introduction

The Enterprise Service Bus (or ESB) is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world. It facilitates mediated interactions between service endpoints. The Enterprise Service Bus supports event-based interactions as well as message exchange for service request handling.

One of the key challenges in the ESB lies in the construction of valid "message flows" between the service requesters and the service providers that perform the required set of mediation operations. Examples of such mediation operations include the routing of the messages to different end-points (e.g. for load balancing), transforming the messages to overcome differences in data schemas or semantics, different kinds of mediations such as splitting or combining messages, verifying security credentials, logging the messages, looking up the service reference using a registry, etc. Typically, a developer or an architect has to design each flow manually taking into account the specific requirements for that flow. This can be quite tedious and difficult when there are multiple options for each operation (such as different ways of transforming or logging the messages). The manual

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 538–544, 2008.

approach to building the flows is also more prone to errors. Another source of difficulty is that the message flows may have to be manually reconstructed when there is a change in the data schema at any of the end-points, or when the requirements in terms of mediation operations change.

In this paper, we propose a methodology for the automated construction of message flows between the requesters and providers. This involves having a reusable set of components and sub-flows that perform different kinds of mediation operations. These components and sub-flows can be combined together and parameterized in different ways depending on high-level mediation requirements for the final flow. The automated composition allows the user to specify the set of high-level operations and have a message flow be generated automatically.

The key technology behind the automated composition of the message flows is an efficient AI Planning-based approach that constructs the flows given high-level goals specified by an enterprise architect or administrator. Our AI Planner [1] uses a tag-based model of the links in the message flow, the different components and of the goals. In this tag-based model, the inputs and outputs of different components are associated with semantic metadata in the form of a set of tags (or keywords) that are drawn from a taxonomy of tags. This taxonomy can, in fact, be a folksonomy that is built through the collaborative efforts of different developers and architects. Similarly, the high-level goals for the composition can also be described as a set of tags. An example high-level goal may consist of the following tags: RequestLogging, ServiceProxy, ServiceX, RegistryY, AccessControl. These tags together represent a requirement for a flow from a requester to a ServiceX that goes through a proxy which looks up the reference (or address) of the service in a RegistryY. The flow should also log all request messages and verify that that the requester has access to the service. Our planner can take this goal, along with tag-based descriptions of different components, to compose a flow that performs these different mediation operations. In some cases, it may come up with multiple flows, and then it uses a provided metric to rank the different plans (such as the resource utilization of the plan).

The automated construction of flows can be used either in the design phase where a developer is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changing requirements. In this paper, we describe a case study with a set of components that can be composed into different flow. We also provide performance results for our planner in this domain.

2 Message Flows in the Enterprise Service Bus

In our work, we model message flows between service providers and requesters as directed acyclic graphs (DAGs), where the vertices represent different components and the edges represent dataflow links. Each component performs some kind of mediation operation. Request messages flow from the requesters to the providers, and response messages flow in the reverse direction. Note that the

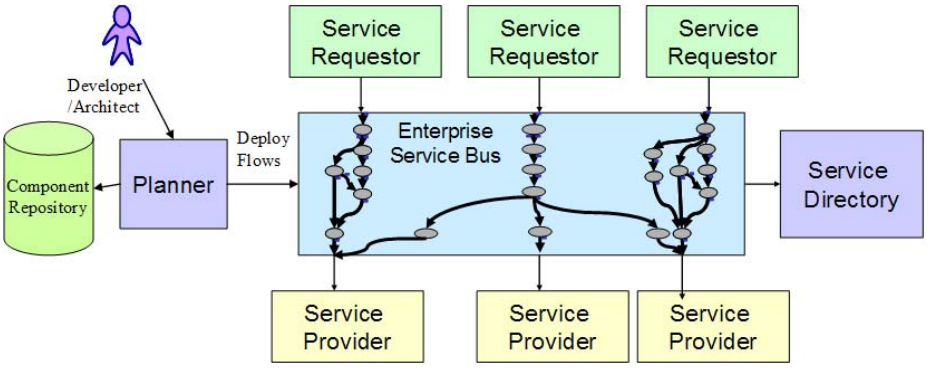

Fig. 1. Architecture for Automatic Composition

request and the response message flows can have very different structures, with different components performing different mediation operations.

Fomally, a flow is a graph $G(V, E)$ where G is a DAG (Directed Acyclic Graph). Each vertex $v_i \in V$ is a component. Each edge (u, v) represents a logical flow of messages from u to v. If there is an edge (u, v), then it means that an output message produced by u is sent as an input message to v. For now, we assume that a vertex has only one incoming edge to it.

Figure 1 shows the architectural elements supporting automatic composition. The planner automatically composes the message flows given high-level goals in the form of sets of tags. These goals may be provided by developers or architects when they are composing new flows between endpoints in the ESB. The planner may also be invoked during runtime, when the goals (or requirements) of existing flows change, and these existing flows need to be replaced by new flows that obey the new requirements.

The planner obtains tag-based descriptions of different components from a component repository. Note that this component repository consists only of mediation components that can be deployed in the ESB. This repository is different from a Service Directory that stores references to service end-points. The Service Directory may be used by components in the ESB to look up service providers that satisfy certain properties.

3 Tag-Based Model of Components and Goals

Let $T = \{t_1, t_2, \ldots, t_k\}$ be the set of tags in our system. A tag hierarchy, H, is a directed acyclic graph (DAG) where the vertices are the tags, and the edges represent "sub-tag" relationships. It is defined as $H = (T, S)$, where T is the set of tags and $S \subseteq T \times T$ is the set of sub-tag relationships. If a tag $t_1 \in T$ is a sub-tag of $t_2 \in T$, denoted $t_1 \prec t_2$, then all resources annotated by t_1 can also be annotated by t_2. An example of a sub-tag relationship is LoadBalancingSelection $\prec$ ServiceSelection. For convenience, we assume that $\forall t \in T, t \prec t$.

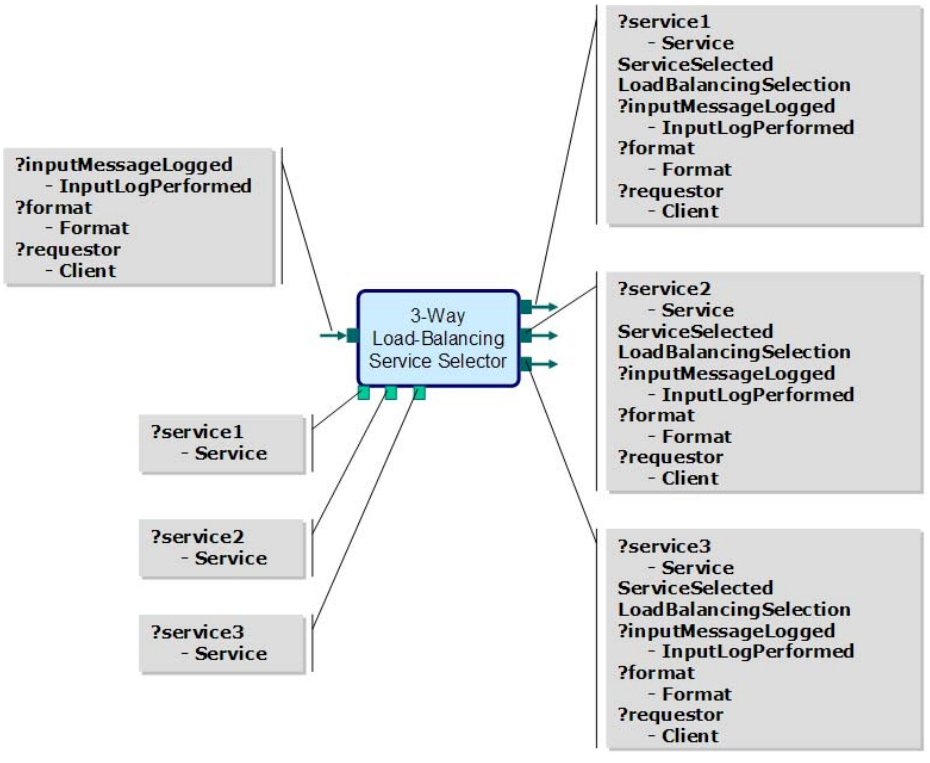

Fig. 2. Tag-Based description of a 3-way Load-balancing Service Selector component. The component has one input port where it receives messages. It forwards this message onto one of 3 output ports, each of which is connected to a possibly different service provider. The identities of these 3 service providers is determined by the values of the parameters (?service1, ?service2, ?service3). All variables in the input, parameters and outputs are associated with their type.

In our approach, tags are used to describe each data link in a message flow is associated with a set of tags. The tags describe the semantics of the messages that flow in the link, as well the actual syntax (using tags that correspond to names of types).

3.1 Component Model

Our model uses the tags in a folksonomy to associate format and semantic information with the input message requirements, the configuration parameters and the output messages of components. Our model also includes the use of variables to describe how the semantic properties of the data are propagated from the input and configuration parametersto the output message. This helps in capturing the notion of semantic propagation, i.e. the semantic description of the output of a component depends on the semantics of the input.

Figure 2 provides an example description of the SelectService component that selects one of 3 service providers based on load-balancing requirements. This component has one input port, 3 parameters and 3 output ports.

The folksonomy-based description of the SelectService includes a variable called ?inputMessageLogged, which is defined to be of type InputLogPerformed. The variable ?inputMessageLogged may be bound to any sub-tag of InputLogPerformed (such as InputMessageLogged and InputMessageNotLogged). This is an example of how our model captures semantic propagation; the output packet is annotated by the same input information about whether the input message was logged or not.

3.2 Composition

The tag-based model allows determining whether a data link, produced by some sub-flow, or a parameter value, can be given as input to another component. The syntax and semantics of a data link, a, can be described by a set of tags, $d(a)$. An input message constraint, I is defined as a set of tags and variables. We define that $d(a)$ *matches* an input constraint, I (denoted by $d(a) \sqsubseteq I_o$), iff

1. For each tag in I, there exists a sub-tag that appears in $d(a)$.
2. For each variable in I, there exists a tag in $d(a)$ to which the variable can be bound. Note that variables can be bound to any sub-tag of their types.

After a match is found for each input to a component, the tag-description of the output message of the component is then formed by replacing all the variables in the output description by the tags to which they were bound in the input side. The use of variables allows us to describe how the semantic properties of the data are propagated from the input to the output packet.

3.3 Goals and Planning

A goal is also described as a set of tags. The goal is satisfied by a flow that produces a message flow with a data link that *matches* the goal tags.

In order to compose flows given an end-user goal as a set of tags and the descriptions of components, we use a planner based on the SPPL formalism. SPPL [1] is a variant of PDDL (Planning Domain Definition Language) and is specialized for describing stream-based planning tasks (a stream can be considered to be a special kind of a data link). At a high level, the planner works by checking if a set of links available in the current state can be used to construct an input to a component, and if so, it generates a new data link corresponding to the output. It performs this process recursively and keeps generating new links until it produces one that matches the goal pattern, or until no new unique links can be produced.

4 Case-Study

We developed a prototype implementation of our automatic composition approach running on IBM's Websphere Message Broker. For this purpose, we created tag-based descriptions of 60 different mediation components that were

deployed on the bus. These components could be composed into different kinds of message flows that followed different mediation patterns. Examples of the patterns included a service proxy pattern (where a proxy component performed a service endpoint lookup in a registry for request messages), a service selector pattern (where a service selector component routed request messages to different implementations of the same service interface) and a service normalizer pattern (where a service selector component routed request messages to different services with different interfaces and transformation components changed the format appropriately). Each of these basic patterns could be enhanced with additional functionalities such as logging of input messages, access control, logging of transformed messages, service lookups in different registries, etc. In our descriptions of the different components, we associated different tags with the different patterns, the different enhanced functionalities as well as different ways of configuring the basic patterns (such as using different service registries or different service selection criteria like load-balancing or content-based routing).

We have a tag-cloud based interface where tags corresponding to composable flows are displayed and selectable by the end-user as part of his goal. The planner may often come up with multiple flows for the same goal (especially if the goal contains few tags and is hence under-constrained). In this case, the lowest cost message flow is displayed to the end-user, although the user can view the alternative flows in the interface. In our setup, each component is associated with a cost, and the cost of a message flow is the sum of the costs of the constituent components. By default, all the components have the same cost; this results in the shortest message flows being shown to the user.

5 Related Work and Conclusion

The most closely related works are in the area of web service composition. Many different kinds of web service models have been proposed in prior work, and these models have been used for discovery and automatic composition. Some of these approaches use ontologies and associated standards such as OWL-S to describe components used in composition [2,3,4]. The key difference in our approach is that message flows in the ESB are generally data processing flows, where the the models of the components must be able to express message mediation and transformation operations, but need not model the state of the component itself. For describing the messages themselves, we use a much simpler tag-based model that reduces the knowledge engineering work required upfront for composition (compared to the more complex models suggested in prior work).

In conclusion, we have described propose an AI Planning-based approach for the automated management and configuration of the ESB. In this approach, message flows between requesters and providers are constructed automatically from high-level goals specified by an enterprise architect or administrator. This automated construction of flows can be used either in the design phase where an architect or developer is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changed requirements.

References

1. Riabov, A., Liu, Z.: Planning for stream processing systems. In: AAAI (2005)
2. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: WWW (2002)
3. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
4. Heflin, J., Munoz-Avila, H.: LCW-based agent planning for the semantic web. In: Ontologies and the Semantic Web, AAAI Workshop (2002)