Representing and Accessing Design Knowledge for Service Integration

Karthikeyan Umapathy  
University of North Florida  
k.umapathy@unf.edu

Sandeep Purao  
Pennsylvania State University  
spurao@ist.psu.edu

Abstract

Process construction from existing services requires use of appropriate design knowledge. For services that are mapped to underlying legacy applications, this takes the form of enterprise integration solutions. Design knowledge in this domain is available in the form of Enterprise Integration Patterns (EIP). These patterns are, however, difficult to understand; they also use primitives that are different from those used for process representation. As a result, accessing EIP based on process requirements remains a cognitively demanding task for designers. In this paper, we describe a knowledge-base that represents the EIPs, infusing them with semantics derived from speech acts; and a set of heuristics, which can be used to retrieve EIPs for a set of requirements. An example serves to illustrate how the two can work in tandem to assist the designer.

1. Introduction

Supporting cross-functional business processes requires building enterprise integration solutions that connect legacy applications wrapped as services. Building enterprise integration solutions, in turn, involves activities such as (a) identification of requirements for integrating underlying system functionalities and (b) exploring appropriate design knowledge that address those requirements [1, 2]. One possible source of knowledge useful for building successful enterprise integration is codified and available as enterprise integration patterns (EIPs) [3].

EIPs represent abstract solutions (much like Gamma’s design patterns [4]), i.e., they do not start as suggested implementation code; instead, they are conceptualized as recurring solutions that designers can use to address categories of integration requirements. The best-developed set of EIPs includes a set of 65 patterns from Hohpe and Woolf [3]. They organize the patterns in seven categories. The first three categories (integration styles, endpoint patterns, system management patterns) suggest different ways of exchanging documents, producing or consuming messages, and managing the performance of messaging systems. The last four (channel patterns, message construction patterns, routing patterns, and transformation patterns) suggest different ways of integrating systems based on how they transport, construct, route, and transform messages.

A designer interested in accessing this knowledge must understand the integration requirements, often represented in the form of business processes, and identify the appropriate EIP(s) based on this understanding [1]. The cognitive effort being performed by the designer for this task is called analogical reasoning [5], which requires an understanding of the mapping between a problem (requirements) and a solution (patterns) [6]. Clearly, this requires knowledge of (a) the integration problem, (b) the patterns (in our case, EIPs), and (3) any constraints on the usage of patterns (in our case, any dependencies among EIPs) [6].

In addition to the above, general cognitive challenges related to use of patterns, there is one other roadblock that is specific to the use of EIPs. It deals with the difference between primitives used for representing integration requirements and those used for representing EIPs. Business processes (as representations of integration requirements) may be developed using Business Process Modeling Notation (BPMN) [7], a de facto standard. BPMN models represent different tasks performed by legacy systems that are logically interlinked to form end-to-end processes [8]. These tasks typically represent the sequencing of business tasks, decision points, and events, i.e., the logic for executing a process – primitives that are very different from those used by EIPs. The primitives used by EIPs, on the other hand, include producing or consuming messages, managing performance of messaging systems, and how they transport, construct, route and transform messages [3]. The identification and retrieval of EIPs, thus, remains a cognitively demanding task (requires mapping abstracted solutions against specific problems), further exacerbated by the mismatch between primitives (used by EIPs and business processes) [2].

In this paper, we describe a knowledge-base that represents the EIPs, infusing them with appropriate semantics that ease cognitive complexity for accessing
EIPs. We also describe a set of heuristics, which can be used to retrieve EIPs for a set of requirements. We use an example to illustrate how the knowledge base and heuristics work in tandem to assist the designer to develop enterprise integration solutions.

2. Structuring the EIP Knowledge-Base

To ease the mismatch between primitives, we argue that it is necessary to infuse additional semantics into the representation of each EIP. Our goal is to make the EIPs more easily accessible as a result of this infusion. The semantics we select must satisfy the dual purpose of: (a) representing messaging primitives that underlie the EIPs, and (b) provide a path to mapping these primitives against the task-based primitives used to represent BPMN models. That is, they should act as an effective intermediary between the two, allowing designers to move back and forth between the EIPs (the solution domain) and the integration requirements (the problem domain).

Of the few candidates available (e.g. conceptual graphs, context-free textual representations, speech acts), the one most directly suited to our purpose is speech acts [9]. The appropriateness of speech acts is directly related to their ability to more richly describe the performance of actions. Speech acts capture the idea of `speaking’ as `acting.’ A well-cited example is the phrase ‘I do,’ which represents an utterance, where the speaker uses language to perform an act, not simply to describe a fact from a universe of discourse.

Speech acts are, therefore, an effective mechanism to describe the accomplishment of tasks and the message exchanges that take place in aid of those actions [10]. Using speech acts to describe EIPs, thus, requires codification of strategies and tactics contained in each EIP in the form of messages that are produced and consumed by systems involved, and order in which the exchange of message occurs [3]. Speech acts provide a way to categorize the different kinds of messages that are produced and consumed, and how they are specified to form a pattern. Based on the order in which the exchange of message occurs, speech acts can be arranged in sequence to represent a pattern. Thus, each EIP can be represented as sequence of speech acts. It is possible, however, that the EIPs provide multiple navigation paths (including decision points). Although this may require, particularly for multi-party interactions, representation formalism such as state-nets, for the EIPs, simpler representation formalism is feasible.

The decision to represent EIPs with speech acts, however, leads to further, inter-related, challenges that must be addressed: (a) deciding on a parsimonious set of speech acts, (b) categorizing messages within the categories provided by this parsimonious set. Although the following presentation describes these as sequential tasks, the work required significant interaction between the two sub-tasks.

We obtained the first (a parsimonious set of speech acts) from an analysis of prior work including Moore’s flexible e-commerce communications [12] and Johannesson and Perjons’s prescriptions for process modeling [10]. A combination of the two suggests an initial set of 9 speech acts, which include: Acknowledge, Cancel, Commit, Direct, Disagree, Fulfill, Inform, Propose, and Query. More details on the analysis as well as definitions of speech acts available elsewhere [2].

We constructed the second (categorizing messages in EIPs with this set of speech acts) by representing kinds of message exchanges supported by each EIP as sequence of speech acts. Figure 1 (see next page) provides two examples along with the rationale for the representation. As an example of the iterative process, we found that two speech acts, Commit and Disagree, were superfluous (i.e. they were subsumed under other speech acts) and were, therefore, dropped during this process.

3. Accessing the EIP Knowledge-Base

Accessing the EIP knowledge base requires identifying integration requirements from the BPMN models. One straightforward heuristic to do this is to identify interactions among participants in a business process [13]. An “interaction” defines a sequence of adjacent tasks that implicates two or more performers [14], including the roles ‘initiator’ and ‘responder’ [15]. This heuristic (change in performer) is useful for detecting and defining an ‘interaction’ in the business process. Speech acts can then be used to describe the actions performed; and the sequences of speech acts can be used to codify the interaction.

Deciding on the appropriate speech act to describe the actions performed, however, requires yet another intermediate mapping. This mapping is provided by a characterization of tasks using “action types” [12]. Action types represent high-level business actions performed by participants through communication directed towards other participant(s) [16].

---

1 An analogy to our use of speech acts in this manner is found in agent communications [11]
Enterprise integration patterns as sequence of speech acts

The complete sequence of mappings that allows access to EIPs based on integration requirements involves following activities:

- Identify interactions in the BPMN models
- Characterize each task using action types
- Map each task against possible speech acts
- Identify EIP using sequence of speech acts

The components that help realize this mapping include:

- a set of action types that depict high-level business actions to categorize tasks in BPMN models, and
- associations between action types and speech acts.

We obtain the first, high-level business actions, to categorize tasks from business activity behaviors described in the Unified Modeling Language (UML) specification [17]. We construct the second by mapping the activities indicated in the action types against intended actions performed through the speech acts. Table 1 lists eleven action types that resulted from this analysis along with associations between action types and speech acts.

<table>
<thead>
<tr>
<th>Action Types</th>
<th>Speech Acts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept with no receipt send</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>Reject with no receipt send</td>
<td>Cancel</td>
</tr>
<tr>
<td>Invocation</td>
<td>Direct</td>
</tr>
<tr>
<td>Declare completion of task</td>
<td>Fulfill</td>
</tr>
<tr>
<td>Accept and send receipt</td>
<td>Inform</td>
</tr>
<tr>
<td>Provide information</td>
<td></td>
</tr>
<tr>
<td>Raise Exception</td>
<td></td>
</tr>
<tr>
<td>Reject and send receipt</td>
<td></td>
</tr>
<tr>
<td>Propose to perform task</td>
<td>Propose</td>
</tr>
<tr>
<td>Request for Information</td>
<td>Query</td>
</tr>
<tr>
<td>Request to cancel task</td>
<td></td>
</tr>
</tbody>
</table>

We developed ontological representations of the speech act-based mechanism using Web Ontology Language (OWL) [21]. OWL provides a standardized semantic mark-up language for publishing and sharing ontologies intended to be processed by applications [21], and supplies the required formalisms to encode hierarchies of classes that describe concepts and relate these classes to each other using properties in a descriptive logic based constraint language [21, 22].

The OWL knowledge base consists of five classes – Action Type, Pattern, Interaction, Speech Act, and Task; and nine properties – hasActionType, hasAssociation, hasSender, hasSenderSA, hasSpeechAct, hasReceiver, hasReceiverSA, hasPattern, and hasPerformer. Figure 2 provides conceptual model for the OWL knowledge base.
Figure 2. Conceptual model of the OWL knowledge base

The cardinality of relationship between the Action Type class and Speech Act through hasSpeechAct property is many-to-one. The Speech Act class consists of 7 sibling classes each capturing specific speech act used for representing typical actions and message exchanges performed in enterprise integration domain. Each Task individuals is also related to maximum one Speech Act class through hasSpeechAct property.

The Interaction class is used for capturing integration requirements, which are represented as interactions between Task individuals. Therefore, each Interaction individuals must be associated to minimum a pair of Task individuals. Set of Task individuals that are initiators of an Interaction individual are captured through hasSender property. Similarly, hasReceiver property is used for capturing set of Task individuals that are responders of an Interaction individual.

Each Interaction instance must be associated with maximum one interaction type which is captured through hasAssociation property. Following are the possible interaction types in a business processes: one to one, one to many, and many to one. Each Interaction individual is also associated with a minimum of one EIP class through hasPattern property.

The EIP class consists of 22 sibling classes, each capturing a specific EIP. Each EIP class is associated to specific set of speech act sequences, captured through hasSenderSA and hasReceiverSA properties. hasSenderSA property captures set of speech acts that represents intended message exchange by initiators of an interaction, while hasReceiverSA property captures set of speech acts that represents intended message exchange by responders.

5. Heuristics to Retrieve EIPs

The knowledge-base is accompanied by a capability to make inferences about appropriate EIPs based on requirements identified from BPMN models. We use the Bossam OWL Reasoner [23] to implement these heuristics. Bossam is a RETE-based forward chaining rule engine with native supports for reasoning with OWL ontologies [24]. The Bossam inference engine translates OWL classes and restrictions as facts, thus, the relationships among action types, speech acts, and EIPs are declared as sets of facts. A set of rules allows inferences about appropriate EIPs based on these declared sets of facts. Below we provide overview of process used for inferring EIP along with heuristics used.

In order to facilitate this inference, requirements, i.e., interactions and their associated business tasks in the given business processes are identified. Then, business tasks are declared as Task individuals along with their action type and performer details as additional sets of facts to the Bossam OWL Reasoner. An example is shown below for declaring a Task individual using syntax required by the Bossam Reasoner:

individual f001 is Task and hasActionType = Accept_with_no_receipt_send, hasPerformer = “Retailer”;

After declaring tasks, requirements are declared as Interaction individuals along with their type of interaction. An example is shown below:

individual f002 is Interaction and hasAssociation=”One_toOne”;

After declaring interactions, senders and receivers of interactions are then declared as facts. The manner in which these are described is shown below:
fact f003 hasSender (f002, f001, “Retailer”);
fact f006 hasReceiver (f005, f004, “WarehouseA”);

After declaring all tasks and interactions involved in the given business process, appropriate rules are added to Bossam OWL Reasoner to aid in inferring process. As a first step, the rule for inferring appropriate speech act for each tasks involved in requirements is added to Bossam Rule set. The rule used for inferring mapping between task and speech acts is given below:

Running the Bossam OWL Reasoner with the new fact and rule sets provides a mapping between identified tasks and appropriate speech act that reflect actions performed by the identified task. Speech act sequences are constructed for each requirements based on speech acts inferred for each identified tasks, which are then, declared as additional sets of facts to the Bossam OWL Reasoner. The speech act sequence is then constructed by assigning set of speech acts that represents intended actions by initiators of the interaction to the hasSenderSA property and similarly actions of responders to the hasReceiverSA property. The rules for capturing speech acts of initiators of an interaction, and for capturing speech acts of responders of an interaction are shown below:

The heuristics capture speech act sequences for each requirement. Speech act sequences for each EIP has been predetermined and captured in OWL knowledge base through hasSenderSA and hasReceiverSA properties for each EIP classes. Thus, Bossam would have already declared speech act sequences for each EIP class as facts. Therefore, we need to write set of rules for each EIP that would map its speech act sequences to that of each requirement. Sample rule set that is used for inferring mapping between the speech act sequences and EIP is given in the table 2.

Currently, we have 27 rules for making inference on appropriate EIP for given set of requirements. First rule associates each task to a specific speech act based on declared action type. Second and third rule constructs sequence of speech acts for each interaction identified. Rest of the rule set are dedicated for identifying appropriate EIP for each interactions based on the sequence of speech acts. Running the Bossam OWL Reasoner with the current fact and rule set would provide mapping between the identified requirement and appropriate set of EIPs.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Inference Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregator</td>
<td>If Interaction ind has Many to One association between initiator y and responder b AND Relevant speech act for action performed by Initiator(s) y is Inform AND Relevant speech act for action performed by Responder b is Inform Then Appropriate pattern for identified Interaction ind is Aggregator</td>
</tr>
<tr>
<td>Command Message</td>
<td>If Interaction ind has One to One association between initiator y and responder b AND Relevant speech act for action performed by Initiator(s) y is Direct Then Appropriate pattern for identified Interaction ind is Command Message</td>
</tr>
<tr>
<td>Point-to-Point Channel</td>
<td>If Interaction ind has One to One association between initiator y and responder b Then Appropriate pattern for identified Interaction ind is Point-to-Point Channel</td>
</tr>
</tbody>
</table>
6. An Illustration

To demonstrate the application of the mapping developed in this research, we use a supply chain management scenario [25] developed by the Web Services Interoperability Organization (WS-I). The specific business process we identified for this demonstration is customer order processing. In this process, a Customer places an order at a retailer’s website, the Retailer requests Warehouses to send stock details for the Customer’s order items. The Retailer then prepares and returns a quote for the Customer’s order (see Figure 3). A number of requirements can be identified in this process (see Table 3). The appropriate speech act is identified for each task in the identified requirements based on their action type. These identified interactions, tasks, and their action types are declared as facts to the Reasoner. Running the Reasoner with the fact set results in a mapping shown in Table 4. For instance, the action type for task 1 is Request for information because the customer is sending a customer order requesting a quote. The associated speech act for the Request for information action type is Query, hence, the Bossam OWL Reasoner mapped task 1 to the Query speech act. Requirement 3 is a many-to-one interaction (4, 5, 6, and 7), with the Inform speech act for initiators and the Acknowledge speech act for responders, therefore, the Bossam OWL Reasoner suggested the Aggregator pattern as appropriate EIP.

Figure 3. Business process diagram for customer order processing

Table 3. Speech act sequence for identified interactions

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Tasks</th>
<th>Performer</th>
<th>Role</th>
<th>Action Type</th>
<th>Speech act</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Customer</td>
<td>Initiator</td>
<td>Request for information</td>
<td>Query</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Retailer</td>
<td>Responder</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Retailer</td>
<td>Initiator</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Warehouse A</td>
<td>Responder</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Warehouse B</td>
<td>Responder</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Warehouse C</td>
<td>Responder</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Warehouse A</td>
<td>Initiator</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Warehouse B</td>
<td>Initiator</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Warehouse C</td>
<td>Initiator</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Retailer</td>
<td>Responder</td>
<td>Accept and no receipt</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>Retailer</td>
<td>Initiator</td>
<td>Provide information</td>
<td>Inform</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Customer</td>
<td>Responder</td>
<td>Accept and no receipt</td>
<td>Acknowledge</td>
</tr>
</tbody>
</table>
### Table 4. Enterprise Integration patterns for identified interactions

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Speech act sequence</th>
<th>Patterns Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Query, Inform</td>
<td>Point-to-Point Channel Document Message</td>
</tr>
<tr>
<td>2</td>
<td>Inform, Inform, Inform, Inform</td>
<td>Recipient List</td>
</tr>
<tr>
<td>3</td>
<td>Inform, Inform, Inform, Acknowledge</td>
<td>Aggregator</td>
</tr>
<tr>
<td>4</td>
<td>Inform, Acknowledge</td>
<td>Point-to-Point Channel Document Message</td>
</tr>
</tbody>
</table>

7. Integrating Services

In order to develop complete integration solutions, the designed solution must provide a path towards implementation. Web services are an ideal platform for implementing enterprise integration solutions because the former is platform-independent. Then, it is essential for designers to provide appropriate means to generate service composition specifications and service conversation specifications based on integration solutions developed.

Service composition specifications such as WS-BPEL, specifies sequence of tasks that must be performed by participating services in order to achieve particular business goals. Ouyang [26] and White [27] provide approaches to generate WS-BPEL specifications based on BPMN models.

Service conversation specifications such as WS-CDL [28] and cpXML [29], describes peer-to-peer interactions between participating services as a multi-step exchange of correlated messages within an explicit conversational context that is established on first contact, maintained for the duration of the conversation, and discarded at the end. Elsewhere, we have provided algorithm for generating service conservations specification based on designed integration solutions using EIP [30].

8. Discussion

In this paper, we have outlined a mechanism that developers can use for service integration. It uses a speech acts based mechanism to access EIPs based on requirements depicted in the BPMN models. In order to implement this mechanism, a knowledge base that captures relationships and constraints among concepts such as BPMN, action types, speech acts and EIPs is developed. We have provided detailed discussion on how OWL is used as knowledge base and heuristics used to make inference on appropriate EIPs for a given integration requirement. We have also demonstrated applicability of our approach through an example.

Elsewhere, we presented a methodology, which designers can use to decompose business processes into interactions, identify appropriate EIP for each identified interaction, and then generate web service solutions [30]. We have developed a software artifact embedding speech act based mechanism, OWL knowledge base, heuristics, and Bossam reasoner described in this paper. Details of the software artifact can be found elsewhere at [30]. As part of future work, we plan to extend the software artifact to include ability to resolve partial and conflicting sets of selected EIPs. We plan to conduct empirical evaluation of the software artifact with experienced designers.

9. References


