Conversation Patterns: Workshop Report

Gregor Hohpe (Editor), Google, Inc.

gregor@hohpe.com

Abstract

In a service-oriented architecture systems communicate by exchanging messages. Message passing provides for robust and loosely coupled interaction but it also provides less structure than traditional RPC models, which are based on a fairly rigid request-response interaction style. Instead, messages exchanged over time between a set of parties can form a multitude of conversations. An expressive contract between communicating parties should define a coordination protocol that describes which conversations are legal. Such a protocol can be expressed in different ways, for example through choreography or public endpoint process definitions. The purpose of conversations patterns is to document common forms of conversations in the design pattern format, highlighting design trade-offs and popular implementations.

As part of the Dagstuhl Seminar 06291 on *The Role of Business Processes in Service Oriented Architectures* in July 2006 we held a Workshop on Conversation Patterns. In this paper, we report on the results of this workshop.

Introduction

Design patterns have established themselves as a valuable design and learning tool in the software development community. Originating from the design and architecture of cities and buildings [APL], design patterns capture knowledge as "mind-sized" chunks in a structured format. The format describes the *context* in which the *problem* occurs, discusses the *forces* affecting any potential *solution*, and provides a good, known solution, followed by implementation guidance. Each design pattern is identified by a name that forms part of a larger *pattern language*.

Many existing design patterns are based on an object-oriented programming model (for example, [GOF], [POSA]) but the technique has been successfully extended to other programming models, such as asynchronous messaging [EIP]. Capturing knowledge as design patterns has proven particularly useful in areas where programming models are less familiar and design choices are manifold.

Services, the elements of a service-oriented architecture, interact through the exchange of messages. A *conversation* is a set of related messaged exchanged by multiple interacting parties, that each play a defined *role* in the conversation (see also [WEB]). A very simple conversation may consist of one message sent from a service consumer, followed by a response message sent by the service provider. This conversation defines two roles, consumer and provider, and two messages. The rules of the conversation state that the conversation begins with a single request message, which is always followed by a single response message.

Describing the rules of a conversation has been the subject of a multitude of specifications, such as WSCL (Web Services Conversation Language), WSCI (Web Service Choreography Interface), WS-CDL (Web Services Choreography Description Language), BPML (Business Process Modeling Language), and WS-BPEL (Business Process Execution Language). Each specification defines a structured language that expresses the relationship between individual messages that

Dagstuhl Seminar Proceedings 06291
The Role of Business Processes in Service Oriented Architectures
http://drops.dagstuhl.de/opus/volltexte/2006/828

make up the conversation, for example whether messages are required or optional, or specific ordering constraints between messages. The most relevant (and to some extend competing) specifications today are WS-CDL and WS-BPEL. WS-CDL describes the overall choreography that covers the message exchange between multiple participants from the viewpoint of a neutral observer. WS-BPEL on the other hand defines an abstract process template that governs the rules of the message exchange by embedding *send* and *receive* elements inside a process definition. The BPMN (Business Process Modeling Notation) expresses conversations similarly by defining partner processes along with the messages exchanged between them.

The multitude of standardization efforts highlights the fact that describing conversations in a precise and intuitive manner is not trivial. Conversation share many challenges with traditional protocol design, such as avoiding deadlock or livelock situations. Many application programmers are less familiar with these kinds of issues, making conversation design a tedious and often error-prone activity. Competing specifications also make it difficult for developers to settle on an implementation language. The fact that none of the aforementioned specifications include a visual notation, often forces developers to communicate their conversation designs in slightly cryptic XML files or very informal sketches.

To allow a broader set of developer to successfully define conversations without having to dive into the details of one of the specification languages is the goal of a catalog of conversation patterns. A sample of the conversation patterns identified and cataloged at this time is given in [DAG].

Workshop Goals

The workshop was organized around the following goals:

- **Notation**: can we find a visually intuitive notation that is expressive and precise?
- **Categorize patterns**: how can we group individual patterns into categories?
- **Composability**: can individual conversation patterns be combined into larger patterns? Do we need a formal description of such composition?
- **Discover patterns**: brainstorm on new pattern candidates that participants have observed in practice.
- **Related work**: identify related works and highlight commonalities and differences. Identify opportunities for collaboration.
- **Formalisms**: Assess the value of a precise (formal?) representation of the patterns in addition to the purely textual description.

As this is a fairly ambitious list of goals for a two-hour workshop we were not able to address all topics in equal depth.

Notation

Depicting conversations in a visually expressive, precise and intuitive form is not easy. As mentioned, the standardization efforts (with the exception of BPMN) do not include any notation. During our workshop we discussed the following options:

UML 1.x sequence diagrams – these diagrams are familiar to most developers and reasonably visually expressive. However, these diagrams cannot express the coordination protocol, i.e., the rules of the conversation; they simply represent a single example conversation. Therefore, multiple sequence diagrams would be needed to enumerate valid conversations.

UML 2.0 sequence diagrams – UML 2.0 introduced additional design elements into the sequence diagram, namely the concept of a frame. Frames can be used to indicate alternatives, parallel execution, and other concepts. As a result, they care able to express the rules of the coordination protocol that governs which conversations are legal. However, the notation is not particularly visually expressive or intuitive.

Icons – Part of [EIP]'s success is due to the icon language that accompanies the pattern language. The icons are visually expressive, intuitive and are easily incorporated in diagramming tools such as Visio. Wil v.d. Aalst showed a paper that incorporates the concept of an iconic visual notation [AALST].

Sketch – [APL] uses a sketch to illustrate the gist of each pattern. The sketch is intentionally imprecise – it is not a blueprint but rather a rough drawing that focuses on the essence of the pattern rather than its detailed implementation. A sketch is a useful tool to allow people to "see" the pattern without having to look at a detailed drawing.

Temporal Logic: not a graphical notation, but a mathematical way to describe the rules that conversations must abide by.

No clear winner emerged at this stage of the discussion. We concluded that the choice of a suitable notation very much depends on the target audience. We also highlighted the difference between the pattern itself and example implementations. Many more precise notations or languages such as BPEL or WS-CDL are more suitable to express example implementations of the pattern while a sketch is more suitable to convey the spirit of the pattern itself.

For the remainder of the workshop we used basic sequence diagrams. We depicted message run-times where they were relevant for the discussion. In cases where the semantics of a basic sequence diagram were insufficient we added plain English descriptions. Because we favor sketches for the patterns description we include the actual sketches from the discussion in this paper.

As a working model we chose the following candidate notations for the individual pattern elements:

Intent: plain English

Solution: Sketch, Plain English, ASM, Pseudo Code

Example: BPEL, CDL, CPN, Java, C#, Sequence Diagram

Discovering Patterns

We had a brief brainstorm on frequently occurring conversation patterns.

Canonical Example – Placing an Order

We started with a common example: placing an order. The conversation consists of 2 roles (buyer and seller), and 4 messages (see Figure 1):

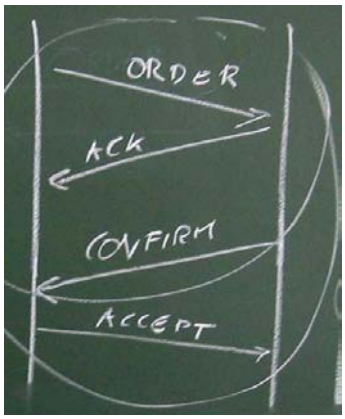

Figure 1: Placing an Order

1. The buyer sends an *Order* message.
2. The seller responds with an immediate *Order Acknowledgment*.
3. Once the details of the order have been confirmed, such as availability of goods, the seller sends a *Confirmation* message.
4. The buyer sends a message to *Accept* the order confirmation.

This example highlights the limitations of sequence diagrams: each diagram only represents a single conversation instance. For example, the buyer could also cancel the order instead of accepting it. However, the sequence diagram only illustrates the "happy day" scenario.

A pattern is not simply a solution, but a solution that addresses a frequently occurring problem in a balance manner. Therefore, it is useful to decompose this conversation into which specific problems (or requirements) it addresses. Sometimes we call this "playing jeopardy" because we know the answer and are looking for the question.

In our case, we decomposed the pattern into the following drivers:

- **Ability to Cancel in Flight**. The buyer has the ability to cancel an order before the confirmation message is sent (see the next section for a more detailed discussion)
- **Quick Acknowledgement**: It is desirable for a buyer (or any service consumer) to receive a quick acknowledgment that informs the buyer that the order has been accepted for processing. This message typically also contains additional information, such as a Correlation Identifier [EIP] that can be used to refer to the same conversation in subsequent messages. The Quick Acknowledgment pattern is used for example by many on-line stores such as Amazon, that send an immediate confirmation with an order number but defer slow checks such as stock availability or credit card validity and send additional messages in case something goes wrong.
- **Nonrepudiation:** ensures that a contract (such as an Order) cannot later be denied by one of the parties involved. This is the reason that the conversation requires the buyer to accept the order after it has been confirmed by the seller. It is important to note that nonrepudiation is not automatically guaranteed by using reliable messaging. For example, someone could have forged an order message on behalf of another buyer.

Special Cases – "Check is in the Mail"

After discussing the "happy day" scenario, we briefly discussed other scenarios than can take place. Because conversations occur between independent processes, parallel execution is not the exception but more likely the norm. This effect is amplified by the fact that message travel times can be significant (indicated by diagonal lines in our sequence sketches).

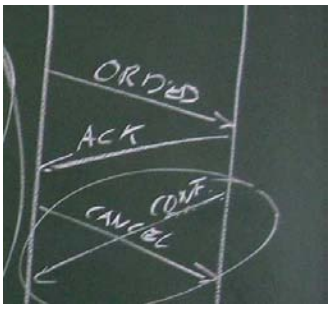

Figure 2: Messages Crossing

Figure 2 shows a common occurrence of what is often referred to as "CrissCross" or "Check is in the Mail"[1]. It describes a situation where two participants send messages to each other and receive each other's message after the send is complete. For example, a buyer might send a Cancel message at the same time the seller is sending a Confirmation message. In this case the coordination protocol has to define what assumptions each participant can make about the state of the conversation. For example, if the coordination protocol defines the state of the order at this point as cancelled, the buyer has to be prepared to receive (and ignore) the Confirmation message as the order is being cancelled. The seller also has to be prepared that an order can be cancelled after a Confirmation message was sent.

Variations from Real Life

We went on to discuss variations of the basic order scenario.

Hard Sell

This pattern describes a situation where the seller "floods" the buyer with multiple quotes in response to a request for quote (RFQ, see Figure 3), possibly even after the buyer has acknowledged a quote.

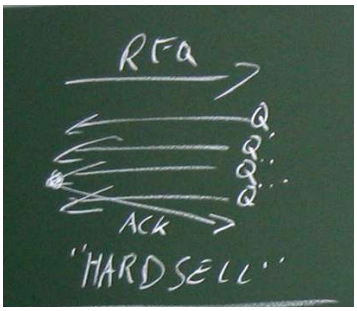

Figure 3: Hard Sell

Counter Offer / Bazaar

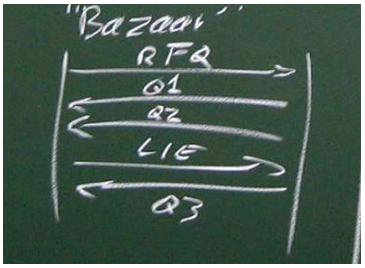

Figure 4 : Bazaar

This scenario describes a situation where the buyer obtains more than one quote from multiple vendors (see Figure 4). What makes this pattern more interesting than a Scatter-Gather [EIP] is that the buyer uses the price quote given by one vendor to negotiate with another vendor. This process can continue in multiple iterations. The other vendor cannot verify that the quote is real, it might well be something the buyer made up (indicated by the Lie message in Figure 4).

[1] This metaphor derives from the common case that someone receives an overdue notice or phone call after they already sent an outstanding payment. Most warning notices actually have a footnote to the user to ignore the notice in case payment has already been submitted.

Composability

After reviewing a few pattern candidates it becomes apparent that larger patterns are composed of similar elements or even smaller patterns. Therefore, it might be useful to define a base vocabulary that helps describe more complex patterns. This base vocabulary might include terms like initiating message, request message, response message, request-response message. It was also deemed useful to define a metamodel that would include the basic elements and their relationships.

One can think of various ways in which patterns can be composed. A larger pattern could be composed of a sequence of two smaller patterns, or one pattern can be inserted into a larger conversation as a sub-conversation.

However, we felt that a more causal; definition is sufficient in the beginning while we are still collecting more use cases and gain experience in the use of the patterns. Especially prototype implementations help validate the patterns. Once the patterns are validated and refined, a more formal metamodel will be more meaningful.

Related Works

During the discussion we identified a number of related works, which we describe here only briefly.

Andries v. Dijk has documented a number of conversation patterns related to contract negotiations [CON].

Alistair Barros, Marlon Dumas and Arthur ter Hofstede have cataloged service interaction patterns [SIP] and are hosting them at www.serviceinteraction.com.

Special thanks go to the workshop participants, Wil v.d. Aalst, Christoph Bussler, Egon Börger, Jorge Cardoso, Ivana Trickovic, Kohei Honda, Johannes Klein, Schahram Dustdar, Uwe Zdun, and many others.

References

AALST Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware, L. Aldred, W.v.d.Aalst; M. Dumas, and A. ter Hofstede

APL *A Pattern Language*, C. Alexander, Oxford University Press, 1977

CON *Contracting Workflows and Protocol Patterns*, Andries van Dijk, in W.M.P. van der Aalst et al. (Eds.): *BPM 2003*, LNCS 2678, Springer, 2003.

DAG *Conversation Patterns*, G. Hohpe, Dagstuhl Seminar 06291, http://kathrin.dagstuhl.de/files/Materials/06/06291/06291.SWM1.Slides.pdf

EIP *Enterprise Integration Patterns*, G. Hohpe, B. Woolf, Addison-Wesley 2003

GOF *Design Patterns: Elements of Reusable Object-Oriented Software*, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison Wesley, 1995

POSA *Pattern-oriented Software Architecture: A System of Patterns*, F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, M. Stal, Wiley, 1996

SIP *Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection*. A. Barros, M. Dumas and A. ter Hofstede: Technical Report FIT-TR-2005-02, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, March 2005.

WEB *Web Services*, G. Alonso, F. Casati, H. Kuno, V. Machiraju, Springer Verlag, 2004

WS-BPEL *Business Process Execution Language*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

WS-CDL *Web Services Choreography Description Language*, http://www.w3.org/TR/ws-cdl-10/

WSCL *Web Services Conversation Language*, http://www.w3.org/TR/wscl10/

WSCI *Web Services Choreography Interface*, http://www.w3.org/TR/wsci/