Cardoso, J., "About the Data-Flow Complexity of Web Processes", 6th International Workshop on Business Process Modeling, Development, and Support: Business Processes and Support Systems: Design for Flexibility. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), 13-17 June 2005, Porto, Portugal, pp. 67-74. ISBN: 972-752-077-4

About the Data-Flow Complexity of Web Processes

Jorge Cardoso

Department of Mathematics and Engineering
University of Madeira, 9050-390 Funchal, Portugal
jcardoso@uma.pt

Abstract. Organizations are increasingly faced with the challenge of managing e-commerce and e-business applications involving Web services and Web processes. In some cases, Web processes' design can be highly complex, due, for example, to the vast number of services carried out in global markets. High complexity in a process has several undesirable drawbacks, it may result in bad understandability, more errors, defects, and exceptions leading processes to need more time to develop, test, and maintain. Therefore, excessive complexity should be avoided. Flexibility and complexity are guiding principles in the design of business processes. In most business processes, flexibility and complexity are related inversely. Processes with a high complexity tend be less flexible, since it is more complicated to make changes to the process. In our previous work we have defined a metric to measure the control-flow complexity of Web processes. The major goal of this paper is to study the issues and establish the requirements for the development of a metric to analyze the data-flow complexity of Web processes.

1 Introduction

In a competitive e-commerce and e-business market, Web processes can span both between enterprises and within the enterprise [1]. While organizations want their Web processes to be simple, modular, easy to understand, easy to maintain and easy to re-engineer, in cross-organizational settings these processes have an inherent complexity.

To achieve an effective process management, one fundamental area of research that needs to be explored is the complexity analysis of Web processes [2]. Studies indicate that 38% of process management solutions will be applied to redesign enterprise-wide processes (source Delphi Group 2002).

Complexity is closely related to flexibility, one of the key enablers of innovation for organizations. Flexibility and complexity are guiding principles in the design of business processes and are in general related inversely. Processes with a low complexity are normally more flexible since they have the capability to quickly change to accommodate new products or services to meet the changing needs of customers and business partners.

In our previous work [3], we have presented a control-flow complexity metric to be used during the development of processes to improve their quality and maintainability. We believe that no holistic metric exist to analyze the complexity of Web processes. Several metrics need to be developed to characterize specific perspectives of Web processes. A control-flow metric alone cannot capture the full complexity of a process. Therefore, an important metric that also need to be investigated is the data-flow complexity.

In this paper we raise a set of questions concerning the development of a data-flow complexity metric for Web processes and propose solutions to some of the challenges.

2 Web process complexity and flexibility

Because flexibility should be a concern during development, the data-flow complexity measures should be considered for use during Web process construction or reengineering to follow complexity trends and maintain predefined flexibility levels. A significant complexity measure increase during testing may be the sign of a brittle, nonflexible or high-risk process. Since processes have the tendency to evolve over time by modification, a process can easily become fragile with age. This compels to use techniques, such as complexity analysis, to assess the system's condition. Complexity metrics can give information concerning the cost and time required to make a given change to a process in order to make it more flexible.

We define Web process complexity as the degree to which a process is difficult to analyze, understand or explain. It may be characterized by the number and intricacy of Web services' interfaces, transitions, conditional and parallel branches, the existence of loops, roles, activity categories, the types of data structures, and other process characteristics [1].

3 Web services and Web processes

Data-flow is the flow of data between the Web services of a Web process. Therefore, we start by giving a brief introduction of Web services and Web processes. A Web service is defined in a standard way using a WSDL (Web Services Description Language) [2] document. A WSDL document contains a set of XML definitions describing Web services using four major elements, which include: input and output messages, data types, port types, and bindings. In our study of data-flow we are interested in messages and types.

While in some cases Web services may be utilized in an isolated form, it is natural to expect that Web services will be integrated as part of Web processes or workflows. The most prominent solution to describe Web processes is BPEL4WS (Process Execution Language for Web Services) [3]. BPEL4WS describes the logic to control and coordinate Web services participating in a process flow. It directly addresses business process challenges such as: control flow (branch, loop, and parallel), manipulation of data between Web services, faults and compensation. The analysis of a BPEL4WS

Web process specification gives a metric for Web service interface integration complexity (section 4.3).

4 Data-flow complexity

The data-flow complexity of a Web process increases with the complexity of its data structures, the number of formal parameters of Web services, and the mappings between Web services' data.

Our data-flow complexity metric is composed of three individual metrics:

a) Data complexity (section 4.1),
b) Interface complexity (section 4.2), and
c) Interface integration complexity (section 4.3).

While the two first metrics are related to static data aspects (data declaration), the third metric is more dynamic in nature and focuses on data dependencies between different activities of a process. We will discuss these three types of data complexity in the following sections.

4.1 Data complexity

One of the greatest challenges in building Web services is creating a common data type system that can be used by a diverse set of programming languages. WSDL does not aim to create a standard for XML data typing. Nonetheless, WSDL uses the W3C XML Schema data type specification [4] as its default choice, currently the most widely used specification for data typing.

As stated previously, we will be studying the different data types present in a WSDL document. More precisely, we are interested in determining the complexity of the data types defined in the W3C XML Schema. The XML schema distinguishes between primitive, derived, and complex data types.

Primitive data types. Primitive data types are those that are not defined in terms of other data types. Currently, XML schema defines 19 primitive data types listed in Table 1. We associate each primitive type with a data complexity constant c_i.

Table 1. XML Schema primitive data types

c_1	String	c_{11}	gYear
c_2	Boolean	c_{12}	gMonthDay
c_3	Decimal	c_{13}	gDay
c_4	Float	c_{14}	gMonth
c_5	Double	c_{15}	hexBinary
c_6	Duration	c_{16}	base64Binary

c_7	dateTime	c_{17}	anyURI
c_8	Time	c_{18}	QName
c_9	Date	c_{19}	NOTATION
c_{10}	gYearMonth		

An example of the declaration of a primitive type is the following:

```
<xs:schema>
…
 <simpleType name="Fahrenheit">
  <xs:restriction base="xs:decimal"/>
 </simpleType>
</xs:schema>
```

In this example, the data named ‘Fahrenheit’ is defined as being of type ‘decimal’. One first research question that needs to be answered is what are the values of the various c_i? Two outcomes can be derived from this question:

a) Since all the data types are primitive then we can assume that they all have the same complexity, i.e. $c_1 = c_2 = \ldots = c_{19}$,

b) On the other hand, we can assume that some primitive data types are more complex than others. For example, it would seem reasonable in some cases to have the complexity of the primitive type ‘dateTime’ (c_7) to be greater than the complexity of a ‘Date’ type (c_9), i.e. $|c_7|>|c_9|$, since a ‘Date’ type can be sought as the aggregation of ‘Date’ and ‘Time’.

Question nº1: Do all the primitive data types have the same data complexity? What is the complexity of each data type?

Derived data types. Derived data types are those that are defined in terms of other data types. There are three kinds of derivation: by restriction, by list, and by union. A data type is said to be derived by restriction from another data type when values are specified that serve to constrain its value space and/or its lexical space to a subset of those of its base type. A list data type can be derived from another data type by creating a finite-length sequence of values of its data type. Finally, one data type can be derived from one or more data types by unioning their value spaces.

The XML Schema has 25 derived types build-in. Some examples are: normalizedString, token, language, Name, ID, integer, unsignedByte, and positiveInteger.

Our first intuition is to recognize that a derived data type must have a greater complexity than its base type. This is because when using a derived data type, the designer needs to remember, not only its base type, but also the information associated with the restriction, list, or union. This seems rather intuitive. But how should we treat restrictions, lists, or unions? Is an extension more complex that a list or is the other way around? Answers to these questions need to be explored.

Question nº2: How to calculate the complexity of derived data types?

Complex data types. XML Schema has three compositor elements that allow constructing complex data types from simpler ones: sequence, choice and all. A sequence defines a compound structure in which the data occur in order. The data within a choice are mutually exclusive. However, there may be multiple occurrences of the chosen data. All defines an unordered group.

In the following example, two new data types are defined: 'location' and 'temperature'. The type 'location' is a complex data type with 3 elements: 'country', 'state', and 'city'. The type 'temperature' is also a complex data type with 3 elements: 'location', 'date, and 'value'.

```
<xs:schema>
…
<xs:element name="location">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="country" type="xs:string"/>
   <xs:element name="state" type="xs:string"/>
   <xs:element name="zip" type="xs:decimal"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="temperature">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="location"/>
   <xs:element name="date" type="xs:dateTime"/>
   <xs:element name="value" type="Fahrenheit"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

While some research has already been carried out to study the complexity of data structures, the XML Schema specification introduces new elements (sequence, choice, and all) that require additional work in this field.

Question nº3: How to calculate the complexity of complex data types?

4.2 Interface complexity

Each operation of a Web service has an interface. An interface defines the input and output parameters of each operation. For example, let us analyze the following WSDL specification:

```
<xs:schema>
…
<xs:element name="location">
```

```
  <xs:complexType>
   <xs:sequence>
    <xs:element name="country" type="xs:string"/>
    <xs:element name="state" type="xs:string"/>
    <xs:element name="zip" type="xs:decimal"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
…
</xs:schema>

<interface name="GetTemperature">
 <operation name="ByLocation" safe="true"
 …
  <input messageLabel="In" element="s:location"/>
  <output messageLabel="Out" element="s:temperature"/>
 </operation>
</interface>
```

The interface ‘GetTemperature’ has only one operation ‘ByLocation’. The operation is composed of one input (‘location’) and one output (‘temperature’).

The input and output can be seen as a reference to a list of formal input parameters and a list of formal output parameters, respectively. Therefore, the inputs of the operation are: ‘country’, ‘state’, and ‘city’. We propose that complexity metric IC (Interface Complexity) to describe the complexity of Web service’s interface (1).

$$IC = IE * PS. \tag{1}$$

Where IE is the Interface Entropy and PS is the Parameters Structure. To calculate the IE of the formal input and output parameters of an operation we use the notion of entropy (2) introduced by Shannon [5].

$$IE(x) = -\sum_{i=1}^{n} p(i)\log_2 p(i) \tag{2}$$

Where, n is the distinct number of data types present in an interface and $p(i)$ the probability of the data type i.

When applying the concept of entropy to interfaces, we can inquire about the diversity of data types present. If the entropy of an interface is 0 then all the parameters are of the same data type. If all the parameters occur in equal number then the entropy is 1. In our previous example, the entropy on the input parameter is calculated as follows:

IC(‘string’, ’decimal’) =
= -p(‘string’)*$\log_2 p$ (‘string’)- p (‘decimal’)*$\log_2 p$(‘decimal’)
= - ⅔*$\log_2$ (⅔) - ⅓ *$\log_2$ (⅓)
= -(-0,39)-(-0,53) = 0,92

Interface entropy gives us an evaluation of the difficulty that a designer has to remember the data types of an operation [6]. Of course there is another aspect that needs to be contemplated, the number of input and output parameters that an operation has, i.e., the parameters structure. The more parameters are present in an operation the more difficult is for the designer to remember all of them. This aspect is captured using the following formula (3):

$$PS(o) = |o_{ip}| * |o_{op}| \quad \textbf{(3)}$$

Where $|o_{ip}|$ is the number of parameters of the input structure and $|o_{op}|$ is the number of parameters of output structure of operation *o*.

4.3 Interface integration complexity

Interface integration complexity (ICC) addresses the concept of coupling. Coupling measures the interdependence of Web services operations, i.e. Web services operations connected with a transition to other Web services operations.

In BPEL4WS, the data is passed between the different Web services in an implicit way through the use data containers. The messages held are often those that have been received from Web services or are to be sent to Web services.

The “assign” statement can be used to copy data from one container to another. Each “assign” activity contains one or more “copy” elements; each “copy” element contains a “from” and a “to” element. Here is a simple example,

```
<assign>
 <copy>
  <from container="PO" part="custInfo"/>
  <to container="shippingReq" part="custInfo"/>
 </copy>
</assign>
```

It is assumed that the higher the coupling [7] between Web services, the more difficult it is for a designer to comprehend a given Web process. The integration of Web services’ operations, due to the polarity [8] of their interfaces, forces an output interface to be connected to an input interface. While the output interface of an operation does not need to be fully integrated to other operations’ input interface, the input interface needs to have its interface fully integrated, i.e. in order to work properly all its inputs need to be mapped to an output belonging to one of the interface.

Question nº4: How to calculate the interface integration complexity of BPEL4WS containers?

5 Conclusions

The complexity of Web processes is intuitively connected to effects such as flexibility, understandability, usability, testability, reliability, and maintainability. Therefore, it is important to develop measures to analyze the complexity of processes so that they can be reengineered to reduce their complexity.

Our goal is to analyze the design of Web processes using measurement strategies. We have discussed the issues related to the development of a data-flow complexity metric raising four important questions that need to be explored and answered.

References

1. Cardoso, J., Evaluating Workflows and Web Process Complexity, in Workflow Handbook 2005, L. Fischer, Editor. 2005, Future Strategies Inc.: Lighthouse Point, FL, USA. p. 284.
2. Christensen, E., et al., W3C Web Services Description Language (WSDL). 2001.
3. BPEL4WS, Specification: Business Process Execution Language for Web Services Version 1.1. 2003, IBM.
4. XMLSchema, XML Schema Part 2: Datatypes Second Edition. 2004, W3C Recommendation 28 October 2004.
5. Shannon, C.E., A mathematical theory of communication. Bell System Technical Journal, 1948. 27(July): p. 379-423.
6. Miller, G.A., The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. Psychological Review, 1956. 63: p. 81-97.
7. Card, D.N. and R.L. Glass, Measuring Software Design Quality. 1990, Englewood Cliffs, New Jersey: Prentice Hall.
8. Cardoso, J. and A. Sheth, Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS). 2003. 21(3): p. 191-225.