Accepted for Publication, Software Process: Improvement and Practice Journal, Copyright © 2006 John Wiley & Sons, Ltd.

Complexity Analysis of BPEL Web Processes

Jorge Cardoso

Department of Mathematics and Engineering
University of Madeira, 9050-390 Funchal, Portugal
jcardoso@uma.pt
Phone: +351 291 705 150, Fax: +351 291 705 199

Abstract. Several organizations have already realized the potential of using WS-BEPL, the Process Execution Language for Web Services, to model the behavior of Web services in business processes. WS-BPEL provides a model for describing simple or complex interactions between business partners. In some cases, WS-BPEL process designs can be highly complex, due, for example, to the vast number of Web services carried out in global markets. High complexity in a process has several undesirable drawbacks, it may result in poor understandability, more errors, defects, and exceptions leading to processes requiring more time to be developed, tested and maintained. Therefore, excessive complexity should be avoided. Processes which are highly complex tend be less flexible, since it is more complicated to make changes to the process. The major goal of this paper is to present two metrics to analyze the control-flow complexity (CFC) of WS-BPEL Web processes. The metrics are to be used at design-time to evaluate the complexity of a process design before implementation actually exists.

Keywords. Web services, Web processes, BPEL, Business processes, workflows, complexity.

1 Introduction

In a competitive e-commerce and e-business market, Web processes can span both between enterprises and within enterprises (Sheth, Aalst et al. 1999). A Web process (Cardoso and Sheth 2005) is a process that models complex interactions among organizations and represents the evolution of workflow technology. While workflows invoke tasks and activities, Web processes invoke Web services. The most well-known language to model Web processes is BPEL (WS-BEPL 2005), and the W3C standard to model Web services is WSDL (Christensen, Curbera et al. 2001). 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 effective process management, one fundamental area of research that needs to be explored is the complexity analysis of Web processes (Cardoso 2005). Studies indicate that 38% of process management solutions will be applied to redesigning enterprise-wide processes (source Delphi Group 2002). Recently, a new field of research for processes has emerged. This new field – termed process measurement – presents a set of approaches to the quantification of specific properties of processes. Important properties to analyze include the estimation of complexity, defects, process size, effort of testing, effort of maintenance, understandability, time, resources, and quality of service. Process measurement is still in its infancy and much work has yet to be undertaken.

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, inversely related. 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. Complex Web processes are more prone to errors. For example, in software engineering it has been found that program modules with high complexity indices have a higher frequency of failures (Lanning and Khoshgoftaar 1994). Surprisingly, in spite of the fact that there is a vast literature on software measurement of complexity, Zuse (Zuse 1997) has found hundreds of different software metrics proposed and described, while almost no research on process complexity measurement has yet been carried out. The only significant work that can be mentioned is the cohesion and coupling metric developed to analyze workflows, proposed by Reijers and Vanderfeesten (Reijers and Vanderfeesten 2004).

In our previous work (Cardoso 2005), we have presented a control-flow complexity (CFC) metric to analyze tri-logic workflows (Cardoso and Cravo 2006). The metric was intended to be used during the development of processes to improve their quality and maintainability. Due to the widespread adoption of WS-BEPL (WS-BEPL 2005), more than 30 enactment engines and editing tools have already been developed, we feel however, that it is important to develop complexity metrics to evaluate the complexity of WS-BPEL (or simply BPEL) processes. Since we believe that no holistic metric exists to analyze the complexity of Web processes, we recognize that several metrics need to be developed to characterize specific perspectives of Web processes.

There are three elements that are fundamental for the definition of any measurement: entity, attribute, and metric. The entities involved in our measurements are BPEL processes. A process can be measured according to different attributes. The attribute that we will target and study is the complexity associated with BPEL processes. Attributes such as time, cost, and reliability have already received some attention from researchers (Cardoso, Miller et al. 2004; Cardoso 2005). The metric that we will study is the control-flow complexity.

This paper is structured as follows: Section 2 presents the various perspectives of process complexity. Section 3 gives brief introduction to WSDL Web services and BPEL Web processes. In section 4, we present the metric that we have developed to evaluate the control-flow complexity of BPEL processes. Section 5 presents the related work. It will be seen that while a significant amount of research has been carried out to quantify the complexity of programs in software engineering, the literature and work on complexity analysis for business processes are almost inexistent. Finally, section 6 presents our conclusions.

2 Web process complexity and flexibility

The flexibility of a process is characterized by a ready capability to adapt to new, different, or changing requirements. According to the IEEE Standard Glossary of Software Engineering Terminology (IEEE 1992), "flexibility is the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed." Following this definition, we view flexibility as the ease of change of a process due to modifications in the environment or in initial requirements. For example, in software engineering, it has been suggested that a way to measure flexibility relies on measures that compute the impact of changes in programs (Li and Offutt 1996). Curtis (Curtis 1979) suggests that "…software complexity determines … how much effort will be required to modify program modules to incorporate specific changes." If a process is complex then it may contain a considerable number of complex components, such as switches, flows, whiles or picks. Adding, removing, or changing an activity from a process requires studying a number of particular cases for which the activity may depend on. The study and analysis of these cases is what makes the process difficult to change, and therefore inflexible. In our view, the more cases a process has the more difficult is to change a process. Being able to handle a large number of cases, as the reviewer states, does not make the process flexible, but makes it complete for a particular domain.

In networked supply chains, process flexibility can be classified into three levels (Ferrara, Hayden et al. 2003). These levels of flexibility require the ability to modify and customize processes, and change processes in real time. The basic property of a process is that it is case-based (Aalst 1998). This means that every task is executed for a specific case. Complex processes tend to be less flexible since they support more cases than simple processes and, therefore, having to take into account all the cases makes it difficult to make changes. For example, eligibility referral (Anyanwu, Sheth et al. 2003) and enrollment processes (CAPA 1997) are complex due to the many cases that exists in each process, which are the results of the complex logic in health-care and educational organizations. Because of the many tasks that are required to handle each case, to change such processes requires considering a vast number of cases which makes the adaptation of the process complex.

Because flexibility should be a concern during development, the control-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 us to use techniques, such as complexity analysis, to assess the system's condition. Complexity metrics can provide 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 (Cardoso 2005).

There is no single metric that can be used to measure the complexity of a process. Based on previous work which identified recurring, generic patterns in workflows, namely Workflow Control Patterns (Aalst, Hofstede et al. 2003), Workflow Data Patterns (Russell, Hofstede et al. 2005), and Workflow Resource Patterns (Russell, Aalst et al. 2005) which characterize the range of control-flow, data, and resource constructs that might be encountered when modeling and analyzing workflows, we identify three main complexity perspectives (Figure 1): control-flow complexity, data-flow complexity, and resource complexity. Since we consider that the type, internal structure, and interface of an activity are also important when computing the complexity of a workflow, we also consider the complexity of activities. Activities can have different levels of complexity since they can be classified into four distinct types (Russell, Aalst et al. 2005): atomic, block, multiple-instance and multiple-instance block. While in this paper we will focus on control-flow complexity, we will present the main ideas behind each complexity perspective as well.

Activity complexity. This view on complexity simply calculates the number of activities a process has. While this complexity metric is very simple, it is very important to complement other forms of complexity. The control-flow complexity of a process can be very low while its activity complexity can be very high. For example, a sequential process that has a thousand activities has a control-flow complexity of 0, whereas its activity complexity is 100. This metric was inspired by lines-of-code (LOC) metric used with a significant success rate in software engineering (Jones 1986).

Control-flow complexity. The control-flow behavior of a process is affected by constructs such as splits, joins, loops, and ending and starting points. Splits allow definition of the possible control paths that exist in a process. Joins have a different role;

they express the type of synchronization that should be made at a specific point in the process. A control-flow complexity model needs to take into account the existence of XOR-split/join, OR-split/join, AND-split/join, loops, etc (Cardoso 2005).

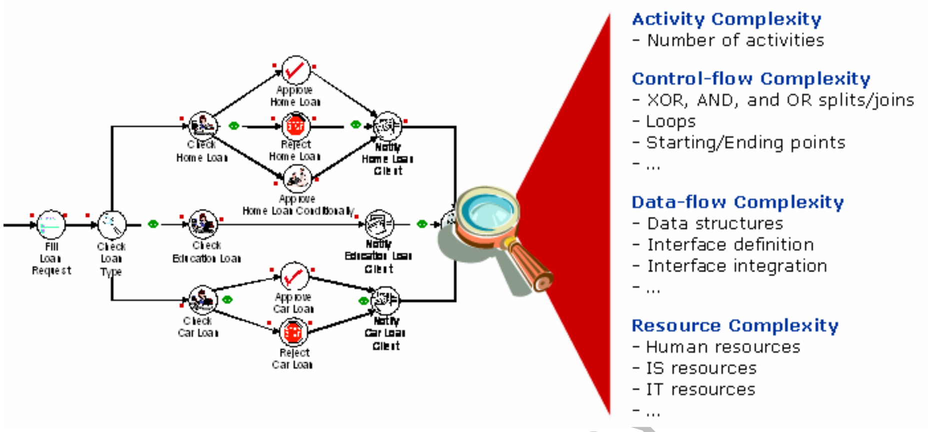

Figure 1. Complexity analysis perspectives

Data-flow complexity. The data-flow complexity of a process increases with the complexity of its data structures, the number of formal parameters of activities, and the mappings between activities' data. A data-flow complexity metric can be composed of several sub-metrics which include: data complexity, interface complexity, and interface integration complexity (Cardoso 2005). While the first two sub-metrics are related to static data aspects (data declaration), the third metric is more dynamic in nature and focuses on data dependencies between the different activities of a process.

Resource complexity. Activities in a process need to access resources during their executions. A resource is defined to be any entity (e.g. human resources, IS resources, and IT resources) required by an activity for its execution, such as a document, a database, a printer, an external application, or role (Du, Davis et al. 1999; zur Mühlen 1999). Resources, such as actors and roles, can be structured in the context of an organization. The structure that is used to shape the different types of resources can be analyzed to determine its complexity. This analysis can help managers to lower administrative costs and better optimize resource utilization.

3 BPEL Web processes

The emergence of e-commerce has changed the foundations of business, forcing managers to rethink their strategies. Organizations are increasingly faced with the challenge of managing e-business systems, Web services, and Web processes. Web services and Web processes promise to ease several current infrastructural challenges, such as data, application, and process integration. With the emergence of Web services, a process management system becomes essential in order to support, manage, and enact Web processes, both between enterprises and within the enterprise (Sheth, Aalst et al. 1999).

A BPEL process is composed of a set of Web services put together to achieve a final goal. As the complexity of a process design increases, it can lead to poor quality and be difficult to reengineer. High complexity in a process may result in limited understandability and more errors, defects, and exceptions leading processes to needing more time to be developed, tested and maintained. Therefore, excessive complexity should be avoided. For instance, critical processes, in which failure can result in the loss of human life, requires a unique approach to development, implementation and management. For this type of process, typically found in healthcare applications (Anyanwu, Sheth et al. 2003), the consequences of failure are very serious. The ability to produce processes of higher quality and less complexity is a matter of endurance.

3.1 Web services

Web services are modular, self-describing, self-contained applications that are accessible over the Internet (Curbera, Nagy et al. 2001). Currently, Web services are described using the Web Services Description Language (Chinnici, Gudgin et al. 2003), which provide operational information. The Web Services Description Language (WSDL) specifies the structure of message components using XML Schema constructs. 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. These elements are illustrated in the following code segment:

```
<message name="getTermRequest">
<part name="term" type="xs:string"/>
</message>
<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

While in this paper we do not evaluate the data-flow complexity of BPEL processes, the study of data-flow complexity involves the analysis of XML Schema data types and the analysis of input and output messages of a Web service (i.e., operations).

3.2 Web processes

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 WS-BPEL (Process Execution Language for Web Services) (WS-BEPL 2005). BPEL provides a language for the formal specification of business processes to facilitate the automated process integration in intra-organization, inter-organization, and the business-to-business settings. BPEL resulted from the combination of WSFL (Leymann 2001) and XLANG (Thatte 2001) languages and uses Web services (Christensen, Curbera et al. 2001; Chinnici, Gudgin et al. 2003) as its external communication mechanism and XML Schema (XMLSchema 2005) for its data model.

BPEL is a XML-based language for describing 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, asynchronous conversations and correlation, long-running nested units of work, faults, and compensation.

A Web process with a high control-flow and data-flow complexity may indicate a higher probability of failure (in software engineering it has been found that program modules with high complexity indices have a higher frequency of failures), increase maintenance costs, and indicate poor understandability.

4 Control-flow complexity

From our perspective there are two independent approaches to develop a CFC metric to analyze business processes: a top-down and a bottom-up approach. The top-down approach starts by formulating a set of general/generic metrics common to various business process languages (such as BPEL, BPMN (BPMN 2005), Meteor (METEOR 2006), etc). These metrics are then applied to specific business process languages to evaluate their applicability and if necessary missing control flow elements can be added to the general metric. In the second approach, i.e. the bottom-up approach, we start by analyzing specific business process languages and formulate specific CFC metrics. Once a reasonable set of business process languages have been analyzed it is then possible to devise general/generic metrics that can be suitably applied to the business process languages analyzed. In our work, we follow the second approach. We already have analyzed and derived specific metrics to three business process modeling languages (Cardoso 2005; Cardoso 2005; Cardoso 2005; Cardoso 2005). A future step will be to aggregate the commonalities of the various different metrics obtained and devise a generic CFC complexity model.

4.1 The CFC metric

A BPEL process definition contains the process logic—the steps that will be followed and outlines which Web services will be executed to achieve a goal or objective. Each step is called an activity. A process always starts with the process element and relates a number of activities. In the process element there has to be one activity specified. In BPEL a process is defined as follows:

```
<process name="process_name">
...
   activity
</process>
```

The formal definition of a process P is P = {a}, where 'a' is an activity. The control-flow complexity of a BPEL process P is simply the complexity of its activity:

$$CFC_{Process}^{BPEL}(P) = CFC_{Act}^{BPEL}(a), a \in P$$

In the following sections we will show how to calculate the complexity of the various types of activities that may be associated with a process.

4.2 Basic activities

BPEL supports two categories of activities: basic and structured activities. Basic activities represent primitive constructs and are used for common tasks. Basic activities can be further classified into three categories: (1) activities for calling and receiving messages from Web services (e.g. <invoke>, <receive>, and <reply>), (2) activities for controlling a process (e.g. <wait> and <terminate>), (3) activities for manipulating data (e.g. <assign>). The behavior of each basic activity is as follows:

- **<invoke>**. Invoking a web service
- **<receive>**. Waiting for the client to invoke the business process through sending a message
- **<reply>**. Generating a response for synchronous operations
- **<assign>**. Manipulating data variables
- **<throw>**. Signaling faults and exceptions
- **<wait>**. Waiting for some time
- **<terminate>**. Terminating the entire process
- **<empty>**. An activity that does not do anything

The <receive>, <pick>, <reply>, and <invoke> activities are called message activities since they communicate with the outside world. As an example, let us see the use of the <receive> activity:

```
<receive
   partnerLink="Registar"
   portType="Registration"
   operation="registerStudent"
   variable="student" />
```

This activity waits for an incoming message. This element allows a business process to do a blocking wait for a particular message to arrive. In our example, the activity is used to wait for the Registrar's office answer after registering a student using the registration system. Since basic activities do not involve an interaction or relationship with other activities, we assign to all of them a complexity value of 1 (one).

$$CFC_{Act}^{BPEL}(a) = 1, a \text{ is a basic activity}$$

In our perspective, assigning a value to a variable, receiving or replying to a message, or terminating a process has the same complexity from a control-flow perspective. This is because the control-flow complexity captures the control-flow in a process and basic activities do not include any control-flow semantics. This fact will become perceptible when we study the complexity of structured activities.

4.2 Structured activities

Structured activities offer a way to structure a BPEL process. Structured activities are more complex and provide simple programmatic control over which steps will be executed in a business process. Structured activities include <case> statements, <while> loops, parallelism constructs such as <flow>, and sequential constructs such as <sequence>. Since the control-flow complexity of each structured activity differs, we need to analyze each activity individually to account for the semantics and particularity of their behavior.

- **<sequence>**. Structures a set of activities to be invoked in an ordered sequence
- **<switch>**. Provides a construct to choose one activity among a collection of activities, i.e. it implements branches
- **<while>**. Defines the notion of loops. This construct executes an activity repeatedly until its associated Boolean condition is no longer true
- **<flow>**. Enables the concurrent execution of activities. It defines a set of activities that will be invoked in parallel
- **<pick>**. Waits on a set of events for one of them to occur and executes a corresponding activity. It allows the selection of one of a number of alternative paths

Structured activities can contain a series of other activities that can be either structured or basic activities. Structured activities prescribe the control-flow of a business process. The control-flow complexity for each structured activity is calculated as follows:

Sequence. A BPEL sequence activity contains a list of activities which are to be executed in lexical order, i.e. the order they are placed within the sequence element (i.e. <sequence>). The sequence activity stops when all activities within it are done.

```
<sequence attributes>
    activity+
</sequence>
```

As illustrated in the previous fragment, a sequence has one or more activity. The formal definition of a sequence S is S = {a_1, a_2, ..., a_n}, where a_i, $i \in \{1,...,n\}$, are activities. The control-flow complexity of a sequence of activities is calculated has follows:

$$CFC_{Act}^{BPEL}(S) = \sum_{a \in S} CFC_{Act}^{BPEL}(a), S \text{ is a sequence}$$

From our viewpoint, the control-flow involved in a sequence of activities is marginal because all the activities are invoked in sequence. Therefore, we express this fact by simply adding the control-flow complexity of the activities of a sequence.

Switch. A BPEL switch structured activity is a construct for introducing conditions based on the evaluation of a Boolean expression. According to (Aalst, Barros et al. 2000), this BPEL construct can be classified as an exclusive choice. The exclusive choice structure defines a point in the process where a certain flow is taken, based on a decision. Most programming languages, such as C, Java, and Perl provide exclusive choice structures. In BPEL the representation of a switch activity is the following:

```
<switch attributes>
    elements
    <case condition="bool-expr">+
        activity
    </case>
    <otherwise>?
        activity
    </otherwise>
</switch>
```

As illustrated in the previous fragment, the <switch> activity consists of an ordered list of conditions specified by a <case> element followed by one optional otherwise element. The activity specified in a <case> element is executed when the Boolean expression associated with the case is true. When none of the cases are true, the activity in the <otherwise> element is executed. The formal definition of a switch Sw is Sw = {a_1, a_2, ..., a_n}, where a_i, $i \in \{1,...,n\}$, are activities. The control-flow complexity of a switch of activities is calculated has follows:

$$CFC_{Act}^{BPEL}(Sw) = n * \sum_{a \in Sw} CFC_{Act}^{BPEL}(a), Sw \text{ is a switch}, n = |Sw|$$

In our perspective, the control-flow introduced by a switch activity is significant. The switch with 'n' conditional activities leads to the exclusive execution of 'n' distinct activities. The control-flow complexity metric for the switch is calculated by multiplying the number of activities in a switch (denoted as $|Sw|$) by the sum of the control-flow complexity of all the activities. The sum of the control-flow complexity of all the activities is multiplied by $|Sw|$ to express the effect that Boolean expression (conditions) have on the complexity of an ordered set of activities. The complexity is linearly dependent on the number of Boolean expressions.

While. The while structured activity offers the possibility to execute an activity in an iterative way. The activity in a while structure is executed as long as the Boolean expression in the condition attribute is true. This construct is also implemented by most programming languages. BPEL represents a while activity in the following way:

```
<while condition="bool-expr" attributes>
    activity
</while>
```

The formal definition of a while W is W = {a}, where ‘a’ is an activity. The control-flow complexity of a while is calculated has follows:

$$CFC_{Act}^{BPEL}(W) = \log_2(CFC_{Act}^{BPEL}(a)+2) * CFC_{Act}^{BPEL}(a),$$
$$W \text{ is a while structured activity, } a \text{ is an activity, } a \in W$$

In our perspective, the control-flow introduced by a while activity is directly dependent on the control-flow complexity of the activity affected by the while. This means that the more activities covered by a while, the greater the complexity. For example, if a while element is applied to a basic activity, such a <invoke>, the control-flow complexity is $\log_2(1+2)*1 = 1{,}6$. This makes sense since the complexity of the basic activity <invoke> is 1 and the <while> increases the overall local complexity. If for example, the while element is applied to a sequence composed of two basic activities, the control-flow complexity is $\log_2(2+2)*2 = 4$. Figure 2 shows a graph that depicts the increase of complexity of a while activity applied to a sequence activity composed of a variable number of basic activities ranging from 1 to 20. The control-flow complexity varies linearly with the number of basic activities in the sequence.

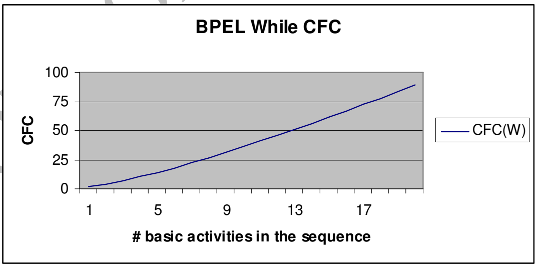

Figure 2. Variation of the CFC of a while activity controlling a sequence activity with basic activities

Figure 2 clearly shows that the complexity of a while structure is linearly dependent on the number of activities that the while covers.

Flow. Concurrency and synchronization of activities is offered by the flow activity. According to (Aalst, Barros et al. 2000), this BPEL construct corresponds to a parallel split. The flow activity enables the creation of splits and joins (Aalst 1998). This pattern defines the structure of a process which is split into several threads of control, all executed concurrently in parallel. The order in which they are processed is not defined.

```
<flow attributes>
    elements
    <links>?
        <link name="ncname"/>+
    </links>
    activity+
</flow>
```

The flow activity offers allows the synchronization (join) of activities within the flow. A flow activity is completed when all its activities are completed. The formal definition of a while F is F = {a_1, a_2, ..., a_n}, where a_i, $i \in \{1,...,n\}$, are activities. The control-flow complexity of a flow activity is calculated as follows:

$$CFC_{Act}^{BPEL}(F) = (n-l)! * \sum_{a \in F} CFC_{act}^{BPEL}(a)$$

F is a flow structured activity, *a* is an activity, *n* =| F |, l = cross boundary links

Since the most fundamental semantic effect of grouping a set of activities in a flow is to enable concurrency, in our perspective, the control-flow introduced by a flow activity is directly dependent on the different ways of arranging the distinct activities in a sequence (permutations). These permutations bring the notion of activity interleaving to our control-complexity metric. For example, if we have a flow structure with two activities (A and B) there exist 2! possible interleaving execution sequences: A followed by B, and B followed by A. Using this rationale, the control-flow complexity of a flow should be $CFC_{Act}^{BPEL}(F) = n! * \sum_{a \in F} CFC_{act}^{BPEL}(a)$, $n=|F|$. Analyzing the semantics of the flow activity, we can examine that BPEL enables the expression of synchronization dependencies between activities using the link construct (<links>). A link construct specifies a dependency between a source activity and target activity as illustrated in Figure 3.

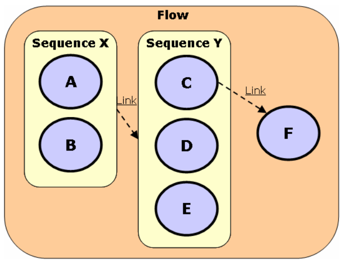

Figure 3. Cross boundary links

The figure shows that links can cross the boundaries of structured activities. There is a link that starts at activity C in sequence Y and ends at activity F, which is directly nested in the enclosing flow. The example also illustrates that sequence X must be performed prior to sequence Y because X is the source of a link named that is targeted at sequence Y. In this example it becomes clear that the rationale followed previously does not hold when links are present. The example demonstrates that having three activities (sequence X, sequence Y, and the basic activity F) we do not have 3! possible interleaving execution sequences. In fact, we only have one sequence, X followed by Y, and Y followed by F. To take into account the existence of links in flow structured activities, we subtract from the number of activities the number of cross boundary links. The number of possible interleaving executions is therefore $(n\text{-}l)!$, where l is the number of cross boundary links of a flow.

Pick. A choice based on information from the outside is offered by the pick activity. Pick provides a construct comparable with a switch activity, except that rather than the decision being based on a Boolean expression, it is based on messages coming (<onMessage>) from a given business partner, or alternately, the expiration of a time-wait object (<onAlarm>) whose expiration will trigger the pick. The <pick> activity specifies that a business process should await the occurrence of one event in a set of events (<onMessage> or <onAlarm> events). The <pick> activity has the following structure:

```
<pick createInstance="yes|no"? attributes>
   elements
   <onMessage partnerLink="ncname" portType="qname"
        operation="ncname" variable="ncname"?>+
     <correlations>?
       <correlation set="ncname" initiate="yes|no"?/>+
     </correlations>
     activity
   </onMessage>
   <onAlarm
        (for="duration-expr" | until="deadline-expr")>*
     activity
   </onAlarm>
</pick>
```

The first event to arrive that is identified in the <pick> completes the pick structured activity. Only one of the activities in the body of the pick takes place. The control-flow complexity of a pick activity is calculated as follows:

$$CFC_{Act}^{BPEL}(Pk) = (2^n - 1) * \sum_{a \in Pk} CFC_{Act}^{BPEL}(a),$$
$$F \text{ is a flow structured activity, } a \text{ is an activity, } n = \#\text{events}$$

From our viewpoint, the control-flow introduced by a pick structured activity is directly dependent on the set of events that may be generated at a given time. This means that the more frequently events are specified in a pick structure, the greater the control-flow complexity of the pick. As we have stated, the pick activity waits the occurrence of one of a set of events and then performs the activity associated with the event that occurred. If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first. A first approach would suggest treating the complexity of the pick activity in the same way as the complexity of the switch activity, since as we have mentioned previously, they are comparable and similar. A closer analysis of the semantics of the pick activity reveals that if several events occur almost simultaneously there is a race and the choice of the activity to be performed is dependent on both timing and implementation of the BPEL process enactment engine. Therefore, to capture the semantics of the pick, we compute all the possibilities of 'n' (n>0) events occurring at the same time. This can be calculated using the power set of the number of events (*n=#events*) specified in the pick, minus one (1), i.e. 2^n-1. We remove one unit since the pick activity is only triggered when at least one event occurs at a given time. The control-flow complexity of a pick structure is dependent on the set *E* of events {e_1, e_2,…, e_n} that the structure can respond to, more precisely on the pick structure dependent on $|\mathcal{P}(E)- \emptyset|$, where $\mathcal{P}$ is the power set.

4.3 Interpretation of CFC values

One important question that needs to be investigated and answered is what the meaning of a given CFC metric is. For example, what is the significance of obtaining a CFC of 16 for a given process? We believe that if organizations and the research community start using our CFC metric it will become apparent that when certain levels of CFC are reached processes become too complex and unmanageable. Our metric was partially inspired in McCabe complexity metric (McCabe 1976; McCabe and Butler 1989; McCabe and Watson 1994) – a well-known and widely used metric – for software engineering. We believe that the interpretation of the CFC will follow a similar path of the one taken by McCabe metric. For example, when using McCabe complexity metric, the limit of 10 indicates a simple program, without much risk. A complexity metric between 11 and 20 designates a more complex program with moderate risk. A metric between 21 and 50 denotes a complex program with high risk. Finally, a complexity metric greater than 50 denotes an untestable program with a very high risk. We expect that limits for the CFC will be obtained and set in the same way, using empirical and practical results from research and from real world use.

5 Scenario

In this section, we describe a scenario to explain and illustrate the need for CFC analysis during the design and aging of a process.

A major bank has realized that to be competitive and efficient it must adopt a new and modern information system infrastructure. Therefore, a first step was taken in that direction with the adoption of a BPMS (Business Process management System) to support its business processes. Since the bank supplies several services to its customers, the adoption of a BPMS has enabled the logic of bank processes to be captured in BPEL schema. As a result, a part of the services available to customers are stored and executed under the supervision of the management system. One of the BPEL schema supplied by the bank is the loan process (process P_1) depicted in Figure 4.

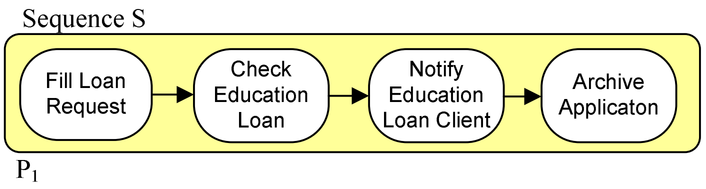

Figure 4. The loan process (version 1)

To represent our process we use the Business Process Modeling Notation (BPMN) (BPMN 2005) since it provides a modeling notation that is easy to use by business process analysts and developers. BPMN can be viewed as a bridge between the modeling and execution of a business process. Particularly, it is possible to generate BPEL execution definitions from BPMN processes.

Process P_1 is very simple and it is composed of only four activities. The Fill Loan Request activity allows clients to request a loan from the bank. In this step, the client is asked to fill in an electronic form with personal information and data describing the loan being requested. The second activity, Check Educational Loan, determines if the loan request should be accepted or rejected. When the result of a loan application is known, it is e-mailed to the client using the Notify Educational Loan Client activity. Finally, the Archive Application activity creates a report and stores the loan application data in a database record.

As this first business process gains acceptance within the bank, since it improves service to customers at several levels, allows significant cost savings, and improves communication amongst employees, the managers of the bank decided to add more services to be supported by the loan process. It was decided to support, not only educational loans, but also home and car loans.

Before making any changes to the process, a control-flow complexity analysis is carried out. To make our example simpler, we assume that each activity in the process is a BPEL basic activity of the type invoke or receive (see section 4.1). The outcome of the analysis indicates that the process has a very low complexity, i.e. 4,

$$CFC_{Process}^{BPEL}(P_1) = \sum_{a \in S} CFC_{Act}^{BPEL}(a) = 4 * CFC_{Act}^{BPEL}(a) = 4, S \textit{ is a sequence}$$

Processes with a low complexity have the capability to quickly change to accommodate new products or services to meet the changing needs of customers and business partners. Based on the complexity analysis results, the process was changed (the new version is reference P_2), having now the structure illustrated in Figure 5.

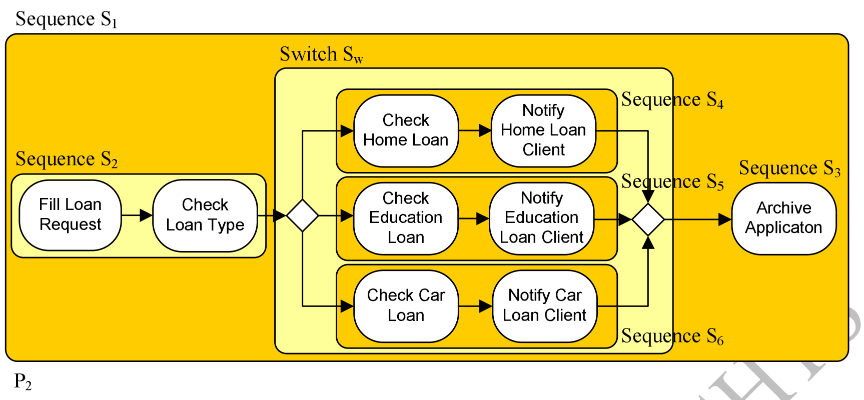

Figure 5. The loan process (version 2)

The new process (version 2) is composed of nine activities. Because complexity was a concern during the development of the new process it still maintains a complexity which is within an acceptable range. The outcome of the CFC analysis indicates that the process has a complexity of 21,

$$
\begin{aligned}
CFC^{BPEL}_{Process}(P_2) &= \\
&= \sum_{a\in S_1} CFC^{BPEL}_{Act}(a) \\
&= \sum_{a\in S_2} CFC^{BPEL}_{Act}(a) + CFC^{BPEL}_{Act}(Sw) + \sum_{a\in S_3} CFC^{BPEL}_{Act}(a) \\
&= 2 + CFC^{BPEL}_{Act}(Sw) + 1 \\
&= 2 + 3*(\sum_{a\in S_4} CFC^{BPEL}_{Act}(a) + \sum_{a\in S_5} CFC^{BPEL}_{Act}(a) + \sum_{a\in S_6} CFC^{BPEL}_{Act}(a)) + 1 \\
&= 2 + 3*(2+2+2) + 1 = 21
\end{aligned}
$$

For the twelve months that followed the design and implementation of the second version of the process several small changes have been introduced to the process. Unfortunately, since the changes were done incrementally and each one had a small impact to the structure of the process, complexity analysis was not carried out during the process redesign. As a result, the process' structure is the following (Figure 6).

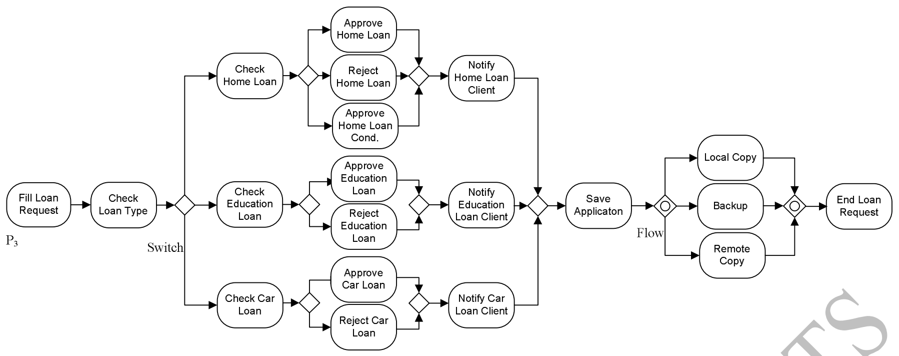

Figure 6. The loan process (version 3)

The process has evolved over time by modification and may have become fragile with age. Therefore, it is necessary to use techniques, such as complexity analysis, to assess the system's condition. A high complexity may be the sign of a brittle, non-flexible, or high-risk process. The outcome of the CFC analysis indicates that the process has a complexity of 49, i.e. 2+3*(5+4+4)+1+3!+1. Since a high complexity was identified the process needs to be reengineered to reduce its complexity.

As another example, let us consider that we have a process a flow element with 4 outgoing transitions. One solution to reduce the complexity of the process would be to replace the flow element with a switch element with two outgoing transitions, where each outgoing transition has a flow element with 2 outgoing transitions. The control-flow complexity would drop from 24 (i.e., 4!) to 4 (i.e., 2*2!). Of course making this reengineering undertaking would depend on the business process under analysis.

6 Related work

While a significant amount of research on the complexity of software programs has been done in the area of software engineering, the work found in the literature on complexity analysis for Web processes and workflows is almost inexistent. Research in software engineering has produced various measurements for software. Among others are lines-of-code (Park 1992), Halstead's measure (Halstead 1977), McCabe's measure (McCabe 1977), the COCOMO model (Boehm 1981) and the Function-Point method (Garmus and Herron 2000). There is a vast literature on software metrics which represents the result of the measurement of the development, operation and maintenance of software in order to supply meaningful and timely management information. Zuse (Zuse 1997) has found hundreds of different software metrics proposed and described for software measurement.

The most important research on complexity analysis for Web processes and workflows can be found in (Reijers and Vanderfeesten 2004; Cardoso 2005; Cardoso 2005; Cardoso 2005; Cardoso 2005). In (Cardoso 2005; Cardoso 2005; Cardoso 2005; Cardoso 2005), a control-flow metric is presented to analyze the complexity of tri-logic workflows (Cardoso and Cravo 2006). The metric is based on the idea of counting the number of new states generated from the introduction of control-flow patterns, such as XOR-splits, OR-Splits, and AND-splits in a process. In (Cardoso 2005) the importance of analyzing the complexity of processes that model team works, communication and collaboration to accomplish significant work and projects is studied. Workflow management systems are a specific type of systems that can be used to capture collaboration and group works processes and thus supports the creation of teamwork and enable collaboration. In some cases, collaboration and group work processes can become highly complex. Therefore, complexity analysis is important. In (Cardoso 2005) a study is made to evaluate the control-flow complexity measure in terms of properties. Weyuker's properties must be satisfied by any complexity measure to qualify as a good and comprehensive one. In (Cardoso 2005) it is argued that a control-flow metric alone cannot capture the full complexity of a process. Therefore, an important metric that should also be investigated is the data-flow complexity. This contribution raises a set of questions concerning the development of a data-flow complexity metric for Web processes and proposes solutions to some of the challenges. In (Reijers and Vanderfeesten 2004), Reijers and Vanderfeesten propose a cohesion and coupling metric developed to analyze workflows. While their work does not take the viewpoint of complexity analysis, it can be easily reformulated to make cohesion and coupling a specific complexity perspective.

7 Future work

In order to empirically validate the complexity metrics that we have described, experiment need to be carried out. An empirical study is an experiment that compares what we believe to what we observe. Such an experiment will play a fundamental role in our work. Zelkowitz and Wallace (1998) stress the importance of using experimental models for validating metrics. The authors suggest experimentation as a crucial part of the evaluation of new metrics. Validation models mainly deal with data collection, experimentation, and data analysis. The information gathered from observation is analyzed and the results are used in determining the validity of a new metric. In our case, an experiment needs to involve several roles, such as business process managers, business analysts, and process implementers. The data collected from the empirical experiment can then be statistically analyzed to determine the existence, or not, of a correlation between the complexity metrics we propose and the ratings of our subjects.

Another important issue that needs to be investigated and answered is what are both the meaning of a given complexity metric (for example, what is the significance of the control-flow complexity of 16) and the precise number to use as a complexity limit in a process development. This answer will be given from empirical results only when organizations have successfully implemented complexity limits as part of their process development projects. For example, when using McCabe complexity metrics (McCabe 1976), the original limit of 10 indicates a simple program, without much risk, a complexity metric between 11 and 20 designates a more complex program with moderate risk, a metric between 21 and 50 denote a complex program with high risk. Finally, a complexity metric greater than 50 denotes an untestable program with a very high risk.

We expect that limits for control-flow complexity will be obtained and set in the same way, using empirical and practical results from research and from real world implementation.

In order to follow a bottom-up approach to develop a generic metric for process modeling languages, it is necessary to analyze the CFC complexity of additional languages. Only once a reasonable set of business process languages have been analyzed it is then possible to devise general/generic metrics that can be suitably applied to the business process languages analyzed. The most well-known languages that still need to be studied include WSFL (Leymann 2001), BPML (BPML 2004), YAWL (Aalst and Hofstede 2003), and BPMN (BPMN 2005).

8 Conclusions

The complexity of BPEL 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. Since flexibility is the ease with which a process can be modified for use in applications or environments other than those for which it was specifically designed, processes with a low complexity tend to be more flexible.

Our work presents an approach to carrying out BPEL process complexity analysis using measurement strategies. We have discussed the issues related to the development of a control-flow complexity metric. The metrics introduced are worth exploring further, since Web processes and Web services are becoming a reality in e-commerce and e-business activities.

In this paper we propose a control-flow complexity metric to be used during the design of processes. This metric is a design-time measurement. They can be used to evaluate the difficulty of producing a BPEL process design before implementation. When control-flow complexity analysis becomes part of the process development cycle, it has a considerable influence on the design phase of development, leading to further optimized processes. Complexity analysis can also be used in deciding whether to maintain or redesign a process. As known from software engineering, it is a fact that it is cost-effective to fix a defect earlier in the design lifecycle than later. To enable this, we have introduced the first steps to carrying out process complexity analysis.

References

Aalst, W. M. P. v. d. (1998). "The Application of Petri Nets to Workflow Management." The Journal of Circuits, Systems and Computers **8**(1): 21-66.

Aalst, W. M. P. v. d., A. P. Barros, et al. (2000). Advanced Workflow Patterns. Seventh IFCIS International Conference on Cooperative Information Systems.

Aalst, W. M. P. v. d. and A. H. M. t. Hofstede (2003). YAWL: Yet Another Workflow Language (Revised Version). Brisbane, Queensland University of Technology2003.

Aalst, W. M. P. v. d., A. H. M. t. Hofstede, et al. (2003). "Workflow Patterns." Distributed and Parallel Databases **14**(3): 5-51.

Anyanwu, K., A. Sheth, et al. (2003). "Healthcare Enterprise Process Development and Integration." Journal of Research and Practice in Information Technology, Special Issue in Health Knowledge Management **35**(2): 83-98.

Boehm, B. (1981). Software Engineering Economics, Prentice Hall.

BPML (2004). Business Process Modeling Language. **2004**.

BPMN (2005). Business Process Modeling Notation - http://www.bpmn.org/.

CAPA (1997). Course Approval Process Automation (CAPA). Athens, GA., LSDIS Lab, Department of Computer Science, University of Georgia.

Cardoso, J. (2005). About the Complexity of Teamwork and Collaboration Processes. IEEE International Symposium on Applications and the Internet (SAINT 2005), Workshop - Teamware: supporting scalable virtual teams in multi-organizational settings, Trento, Italy, IEEE Computer Society.

Cardoso, J. (2005). 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, Porto, Portugal.

Cardoso, J. (2005). Control-flow Complexity Measurement of Processes and Weyuker's Properties. 6th International Conference on Enformatika, 26-28, October 2005, Budapest, Hungary, International Academy of Sciences.

Cardoso, J. (2005). Evaluating Workflows and Web Process Complexity. Workflow Handbook 2005. L. Fischer. Lighthouse Point, FL, USA, Future Strategies Inc.**:** 284-290.

Cardoso, J. (2005). Evaluating Workflows and Web Process Complexity. Workflow Handbook 2005. L. Fischer. Lighthouse Point, FL, USA, Future Strategies Inc.**:** 284.

Cardoso, J. and C. Cravo (2006). Verifying the logical termination of workflows (accepted for publication). 5th Annual Hawaii International Conference on Statistics, Mathematics and Related Fields, Honolulu, Hawaii, USA.

Cardoso, J., J. Miller, et al. (2004). "Modeling Quality of Service for workflows and web service processes." Web Semantics: Science, Services and Agents on the World Wide Web Journal **1**(3): 281-308.

Cardoso, J. and A. P. Sheth (2005). Introduction to Semantic Web Services and Web Process Composition. Semantic Web Process: powering next generation of processes with Semantics and Web services. J. Cardoso and A. P. Sheth. Heidelberg, Germany, Springer-Verlag. **3387:** 1-13.

Chinnici, R., M. Gudgin, et al. (2003). Web Services Description Language (WSDL) Version 1.2, W3C Working Draft 24, http://www.w3.org/TR/2003/WD-wsdl12-20030124/.

Christensen, E., F. Curbera, et al. (2001). W3C Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl.

Curbera, F., W. Nagy, et al. (2001). Web Services: Why and How. Workshop on Object-Oriented Web Services - OOPSLA 2001, Tampa, Florida, USA.

Curtis, B. (1979). In search of software complexity. Workshop on Qualitative Software Models for Reliability, Complexity and Cost, IEEE Computer Society Press.

Du, W., J. Davis, et al. (1999). Enterprise workflow resource management. International Workshop on Research Issues in Data Engineering, Sydney, Australia.

Ferrara, L., F. Hayden, et al., Eds. (2003). The Networked Supply Chain: Applying Breakthrough BPM Technology to Meet Relentless Customer Demands, J. Ross Publishing.

Garmus, D. and D. Herron (2000). Function Point Analysis: Measurement Practices for Successful Software Projects, Addison Wesley.

Halstead, M. H. (1977). Elements of Software Science, Operating, and Programming Systems Series. New York, NY, Elsevier.

IEEE (1992). IEEE 610, Standard Glossary of Software Engineering Terminology. New York, Institute of Electrical and Electronic Engineers.

Jones, T. C. (1986). Programming Productivity. New York, McGraw-Hill.

Lanning, D. L. and T. M. Khoshgoftaar (1994). "Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty." Computer **27**(9): 35-41.

Leymann, F. (2001). Web Services Flow Language (WSFL 1.0), IBM Corporation.

Li, L. and A. Offutt (1996). Algorithmic analysis of the impact of changes to object-oriented software. Int'l Conf. Software Maintenance-ICSM (1996), Los Alamitos, USA, IEEE Computer Society Press.

McCabe, T. (1976). "A Complexity Measure." IEEE Transactions of Software Engineering **SE-2**(4): 308-320.

McCabe, T. J. (1977). "A Complexity Measure." Transactions on Software Engineering **13**(10): 308-320.

McCabe, T. J. and C. W. Butler (1989). Design Complexity Measurement and Testing. Communications of the ACM. **32:** 1415-1425.

McCabe, T. J. and A. H. Watson (1994). "Software Complexity." Crosstalk, Journal of Defense Software Engineering **7**(12): 5-9.

METEOR (2006). METEOR (Managing End-To-End OpeRations) Project Home Page, LSDIS Lab.

Park, R. E. (1992). Software size measurement: A framework for counting source statements. Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University.

Reijers, H. A. and I. T. P. Vanderfeesten (2004). Cohesion and Coupling Metrics for Workflow Process Design. BPM 2004 (LNCS 3080). J. Desel, B. Pernici and M. Weske. Berlin, Heidelberg, Springer-Verlag. **LNCS 3080:** 290-305.

Russell, N., W. M. P. v. d. Aalst, et al. (2005). Workflow Resource Patterns: Identification, Representation and Tool Support. 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005), Porto, Portugal, Springer 2005.

Russell, N., A. H. M. t. Hofstede, et al. (2005). Workflow Data Patterns: Identification, Representation and Tool Support. 24th International Conference on Conceptual Modeling, Klagenfurt, Austria, Springer.

Sheth, A. P., W. v. d. Aalst, et al. (1999). "Processes Driving the Networked Economy." IEEE Concurrency **7**(3): 18-31.

Thatte, S. (2001). XLANG: Web Services for Business Process Design, Microsoft, Inc.

WS-BEPL (2005). Business Process Execution Language for Web Services, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

XMLSchema (2005). XML Schema, http://www.w3.org/XML/Schema.

Zelkowitz, M. V. and D. R. Wallace (1998). "Experimental Models for Validating Technology." IEEE Computer **31**(5): 23-31.

zur Mühlen, M. (1999). Resource Modeling in Workflow Applications. Workflow Management Conference, Muenster, Germany.

Zuse, H. (1997). A Framework of Software Measurement. Berlin, Walter de Gruyter Inc.