Evolving Use Case Maps as a Scenario and Workflow Description Language

Gunter Mussbacher
SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada
gunterm@site.uottawa.ca

Abstract

Since 1996, the core Use Case Map (UCM) notation has remained remarkably stable. As the structure and intent of workflow and scenario languages are very similar, UCMs have been applied to scenario, workflow, and business process modeling. The recent rise of workflow languages for the description of business processes and web services resulted in a more formal assessment method for such languages based on generic workflow and communication patterns. We present such an assessment for UCMs, thereby measuring the applicability of UCMs for workflow description in particular and scenario descriptions in general and gathering evidence on how to evolve the UCM notation. The results are compared to similar assessments which were carried out for current standards for workflow, business process design, and business process execution languages such as the Business Process Modeling Notation (BPMN), the Business Process Execution Language for Web Services (BPEL4WS), and UML 2.0 Activity Diagrams.

1. Introduction

Use Case Maps (UCMs) [28] are an integral part of the International Telecommunication Union's (ITU) effort to standardize the User Requirements Notation (URN) [29]. As any other language, UCMs have to be reevaluated from time to time in light of new technological development. Over the recent years, many languages and techniques have been suggested for the description of workflows for web services and the composition of web services [34]. The main standard for executing business processes is the Business Process Execution Language for Web Services (BPEL4WS) 1.1 published by the OASIS WSBPEL TC [20]. The major players in this field, Microsoft and IBM, have contributed significantly to this standard and have incorporated it into their web service development environments, BizTalk [17] and WebSphere [14], respectively. A new version, WSBPEL 2.0, was recently released. The main standard for modeling business processes at a higher level than BPEL4WS is the Business Process Modeling Notation (BPMN) 1.0 published by OMG [11]. This notation is more business-oriented than technology-oriented, is better suited for the design, management, and monitoring of business processes, and aims to be more easily understood by all business users. A formal mapping to BPEL4WS bridges the gap between business process design and business process implementation. Another OMG standard used to model high-level business processes is the UML 2.0 Activity Diagrams notation [19].

It is important for the future of UCMs, to understand the strengths and weaknesses of the UCM notation compared to these new languages because of the great similarity of UCMs and these languages. At first glance, UCMs are very well suited to describe workflow. As a general scenario notation, the structure and intent is similar to workflow languages. Furthermore, UCMs have already been used for business process modeling [32]. Until now, a more formal assessment of the applicability of UCMs for workflow description has not been performed. Such an assessment must a) indicate whether UCMs are capable of describing commonly encountered workflow situations, and b) allow UCMs' capabilities to be compared to the main standards BPMN, Activity Diagrams, and BPEL4WS.

An assessment based on generic workflow patterns which have been collected from workflow situations frequently encountered when modeling workflow certainly satisfies the first criteria. As the workflow patterns are not just applicable to workflow descriptions but also to scenarios in general (see the examples in this paper), the assessment gives also an indication on the capabilities of UCMs as a general scenario notation and not just a workflow language. The second criteria is also satisfied as assessments based on workflow patterns have already been conducted for BPMN [33], Activity Diagrams [33][35], and BPEL4WS [34]. Note that this paper compares UCMs with BPEL4WS 1.1

instead of WSBPEL 2.0 because an assessment is only available for the former.

The main goal of this paper is to provide insight into how to evolve UCMs and tool support for UCMs. A UCM notation, capable of supporting more workflow patterns, is also generally a more powerful scenario description language. Therefore and whenever possible, this paper presents an improved UCM notation for each workflow pattern that cannot be modeled with the current UCM notation. The results of the assessment (i.e. the applicability of UCMs for workflow description based on UCMs' support for generic workflow patterns) and the comparison with BPMN, Activity Diagrams, and BPEL4WS provide the basis for such an improvement.

As BPEL4WS is a business process execution language, [34] also assessed the language based on generic communication patterns. UCMs are therefore also evaluated with regards to these communication patterns in order to allow a more comprehensive comparison with BPEL4WS. Note that workflow languages can also be analyzed from data and resource perspectives [30] which are beyond the scope of this paper given the space constraints.

In the remainder of this paper, section 2 provides background on UCMs, web services, BPMN, and the generic workflow and communication patterns. Section 3 first describes the details of the pattern-based approach used for the assessment, and then assesses the support of UCMs for each of the workflow and communication patterns. The section closes with a summary of the assessment and a comparison of the results with similar assessments for BPMN 1.0, UML 2.0 Activity Diagrams, and BPEL4WS 1.1. Furthermore, the requirements for evolving UCMs as a scenario and workflow description language are listed based on the findings of section 3. Finally, section 4 gives a conclusion and identifies future work.

2. Background

2.1. Use Case Maps

Use Case Maps (UCMs) [28] are an integral part of the International Telecommunication Union's (ITU) effort to standardize the User Requirements Notation (URN) [29]. UCMs are a scenario notation best suited for the description of functional requirements and if desired, high-level design. UCMs consist of one or more paths describing the causal flow of behavior of a system (e.g. one or many use cases). Optionally, behavioral aspects are superimposed over components which represent the architectural structure of a system (e.g. classes or packages). UCMs abstract from the details of message exchange and communication infrastructures while still showing the interaction between architectural entities. As UCMs integrate many scenarios and use cases into one combined model of a system, it is possible to reason about undesired interactions between scenarios [3], analyze performance implications [21][24], and drive testing efforts based on UCM specifications [5]. As UCMs show architectural structures, various architectural alternatives can be analyzed [6][7][32]. Over the last decade, UCMs have successfully been used for service-oriented, concurrent, distributed, and reactive systems such as telecommunications systems [2][7], e-commerce systems [4], agent systems [12], operating systems [8], and health information systems [1]. UCMs have also been used for business process modeling [32] and aspect-oriented modeling [18]. Many examples of UCMs can be found in the publications referenced in this paragraph.

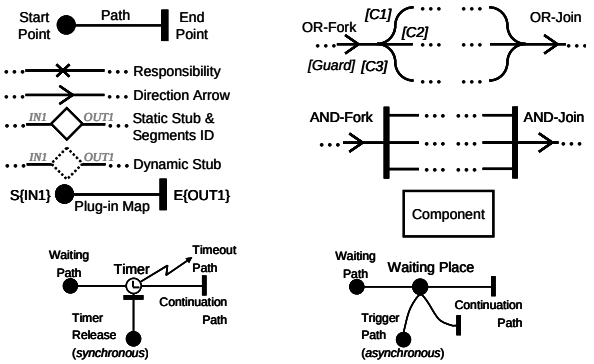

Figure 1: Basic Elements of UCM Notation

The basic elements of the UCM notation are shown in Figure 1. A *map* contains any number of paths and structural elements (*components*). *Responsibilities* describe required actions or steps to fulfill a scenario. *Paths* express causal sequences. *OR-forks* (possibly including guarding conditions) and *OR-joins* are used to show alternatives, while *AND-forks* and *AND-joins* depict concurrency. *Loops* can be modeled implicitly with OR-forks and OR-joins. UCM models can be decomposed using *stubs* which contain sub-maps called *plug-ins*. Plug-in maps are reusable units of behavior and structure. A stub may be *static* which means that it can have at most one plug-in, whereas a *dynamic* stub may have many plug-ins which may be selected at runtime. A *selection policy* decides which plug-ins of a dynamic stub to choose at runtime. Map elements which reside inside a component are said to be *bound* to the component. *Timers* and *waiting places* denote locations on the path where the scenario stops until a

condition is satisfied. If an endpoint is connected to a waiting place or a timer, the stopped scenario continues when this end point is reached (synchronous interaction). Asynchronous, in-passing triggering of waiting places and timers is also possible. A timer may have a *timeout path* which is indicated by a zigzag line. A more complete coverage of the notation elements is available in [9][10][28].

UCMs and UML Activity Diagrams share many characteristics, but UCMs offer more flexibility in how sub-diagrams can be connected, how sub-components can be represented, and how dynamic responsibilities and dynamic components (not shown here) can be used to capture requirements for agent systems. UCMs also integrate a simple data model, performance annotations, and a simple action language used for analysis. Activity Diagrams, however, have better support for data flow modeling and a better integration with the rest of UML. UCMs, on the other hand, are better integrated with goal-oriented models created with the Goal-oriented Requirement Language (GRL) [27]. URN is currently the only requirements notation that explicitly addresses scenarios and goals in one unified language in a graphical way.

jUCMNav [15] is a new Eclipse-based editing tool for UCMs. The tool makes it possible to create, maintain, analyze, and transform UCM models.

A *traversal mechanism* which allows highlighting of individual scenarios and scenario combinations in UCM models is built into jUCMNav. The mechanism is also used when translating UCMs into more concrete design diagrams such as message sequence charts (MSC). Besides improving the usability of the tool and allowing for a smoother transition to downstream modeling activities, the traversal mechanism defines more precisely the semantics of UCMs. There is no standard traversal mechanism for UCMs. The current traversal mechanism is based on a simple but intuitive interpretation of the notational elements for sequences, alternatives, and concurrency in UCMs. An OR-fork is an exclusive or, there is no synchronization on an OR-join, an AND-fork denotes strict concurrency, and stubs are interpreted as containing an exclusive or statement for the selection of a single plug-in map. The current traversal mechanism also allows the definition of Boolean, Integer, and Enumeration variables and is capable of evaluating and changing such *scenario variables* during the traversal of the UCM model.

2.2. Web Services and BPMN

The first era of the Internet allowed for static content to be presented to a worldwide audience. The second era saw the emergence of dynamic content which could be tailored to a single user in order to provide personalized experiences. The Internet has come a long way from its beginnings and is now entering its third era: the programmable web. Web services are at the core of the third era of the Internet. Large corporations such as Microsoft and IBM see web services as the future of the whole information technology [23]. Web services are set to have an enormous impact on e-commerce by making automated business-to-business interactions a reality.

The web services programming stack allows application developers to advertise, locate, and make use of web services [13]. Web services interact over a network based on protocols such as HTTP, FTP, SMTP, or the Internet Inter-ORB Protocol (IIOP). The interface of a web service is described with the Web Services Description Language (WSDL) [31]. Messaging between other applications and a web service occurs with the help of the Simple Object Access Protocol (SOAP) [25]. Universal Description, Discovery, and Integration (UDDI) [26] allows for web services to register themselves and for other applications to locate them. At the top of the stack, and therefore most visible to application developers, are workflow and business process execution languages such as the Business Process Execution Language for Web Services (BPEL4WS) [20] which allow the description and composition of web services.

BPEL4WS, however, is a very technical language which is not well suited for business users. The Business Process Modeling Notation (BPMN) [11] addresses this problem and provides an environment which is more suited for business-oriented users. It allows business processes to be designed, managed, and monitored, and provides a formal mapping to BPEL4WS. Consequently, a good workflow description language can make a very valuable contribution to the future development of web services.

2.3. Workflow and Communication Patterns

In 2000, an analysis of workflow languages resulted in the publication of 21 workflow patterns divided into six groups that describe typical control flow dependencies in workflow models [36]. The first two groups address basic and advanced controls related to branching and merging of control paths. The third discusses structural restrictions on loops and implicit termination. The fourth covers situations with multiple instances of activities. The fifth deals with state-based patterns, while the last group discusses cancellations. See Table 1 in section 3.9 for a complete list of the workflow patterns and sections 3.2 to 3.7 for an explanation of the individual workflow patterns.

Since the initial publication of these patterns, many workflow management systems (e.g. Domino Workflow, FLOWer, I-Flow, MQSeries/Workflow, SAP R/3 Workflow, and Visual Workflow) and standards for business process modeling and workflow modeling (e.g. Activity Diagrams, BPMN, BPEL4WS, BPML, WSCI, WSFL, XLANG, and XPDL) have been assessed based on the collected workflow patterns [36]. Furthermore, a PhD thesis [16] was written "to establish a formal foundation for control-flow aspects of workflow specification languages, that assists in understanding fundamental properties of such languages, in particular their expressive power" [36]. The research culminated in the YAWL (Yet Another Workflow Language) initiative [37] with its goal to create a workflow language with direct support for all of the discovered workflow patterns.

On the other hand, communication patterns [22] have been collected in the context of Enterprise Application Integration (EAI). They are applicable to web services as both, EAI and web services, are concerned with communication flows between distributed processes. The patterns are divided into two groups, synchronous and asynchronous communication. See Table 1 in section 3.9 for a complete list of the communication patterns and section 3.8 for an explanation of the individual communication patterns.

3. Assessment and Evolution of UCMs

3.1. The Approach

We follow the approach in [34] to assess the applicability of UCMs for workflow description. UCMs are analyzed by determining to what extent 21 workflow patterns and six communication patterns are directly supported by the notation. A pattern is directly supported if a UCM language construct exists that concisely expresses the pattern. More complex, workaround solutions are not taken into account since all patterns can be expressed in some way by standard features of specification languages (including those of UCMs). In other words, the mere ability to express a pattern in some way is not sufficient because conciseness and simplicity are key factors to be considered. If current UCM features do not allow workflow patterns to be modeled, new notational elements or a specialized traversal mechanism are suggested to improve the UCM notation (indicated by "new" in the pattern figures).

In general, business processes and their activities can be modeled straightforwardly in UCMs with paths and responsibilities. Business partners can be modeled with components to which certain activities are assigned by placing the relevant portion of the path inside the component. Sections 3.2 to 3.7 go through each of the 21 workflow patterns.

3.2. Basic Control Patterns

Sequence (Figure 2.a)—This pattern is trivially supported through a UCM path.

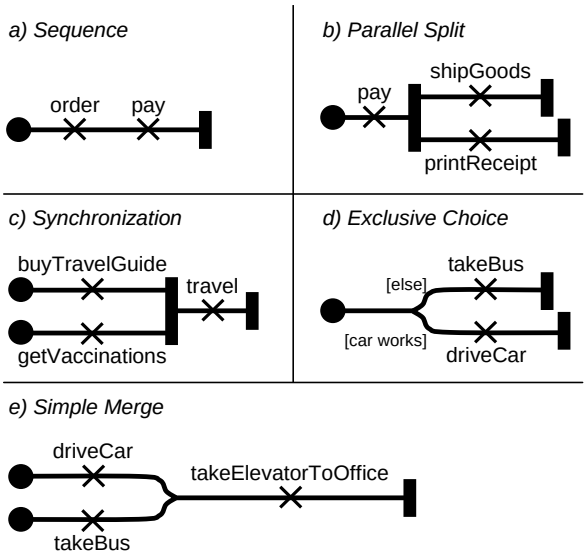

Figure 2: Basic Control Patterns (Group 1)

Parallel Split (Figure 2.b)—This pattern indicates that several activities can be performed in parallel or in any order. The pattern is mapped directly onto an AND-fork which can have any number of parallel branches.

Synchronization (Figure 2.c)—This pattern indicates a point in the workflow where several concurrent branches converge into one single branch. The pattern assumes that each incoming branch is only executed once. The pattern is mapped directly onto an AND-join which can have any number of incoming parallel branches.

Exclusive Choice (Figure 2.d)—This pattern is trivially supported by an OR-fork which can have any number of alternative branches. The conditions for all branches, however, have to be mutually exclusive.

Simple Merge (Figure 2.e)—This pattern describes a point in the workflow where several alternative branches are merged together into one without synchronization. The pattern is directly supported by an OR-join which can have any number of incoming alternative branches. Note that this usage of an OR-join is just a special case of the usage of an OR-join for the Multiple Merge pattern described below.

3.3. Advanced Branching and Synchronization Patterns

Multiple Choice (Figure 3.a)—This pattern describes the situation where more than one alternative branch can be selected and executed at the same time. The pattern is directly supported by an OR-fork which can have any number of alternative branches. Because conditions can be specified unrestrictedly for each branch, multiple branches can be enabled at the same time. Since this pattern deals with concurrency, it may however be more appropriate to extend the UCM notation and allow conditions to be specified on branches of AND-forks. As an alternative, a dynamic stub can also be used. In this case, the selection policy enables multiple plug-ins at the same time. This is the preferred approach if the branches have to be synchronized at any point (see Synchronizing Merge). Even though the pattern is supported by the UCM notation, the current traversal mechanism does not support such an interpretation. Instead, the current traversal mechanism expects exactly one branch of an OR-fork (or one plug-in of a dynamic stub) to be enabled with an option to choose randomly one out of all enabled branches or plug-ins in case of a non-deterministic situation.

Synchronizing Merge (Figure 3.b)—This pattern describes a point in the workflow where several branches are merged together into one with synchronization. The pattern assumes that each incoming branch is at the most executed once. It can be assumed that the number of branches taken is known at the time the merge is reached (at least one branch and at the most all branches). This pattern cannot be modeled with an OR-join (because OR-joins do not synchronize) or an AND-join (because AND-joins require all incoming branches to be taken in order to proceed). The pattern can be modeled with a dynamic stub and one plug-in for each branch if a specialized traversal mechanism can identify a stub with Synchronizing Merge behavior (the default for stubs is Simple Merge). The selection policy of the *synchronizing stub* enables the required number of plug-ins. The traversal mechanism then waits until all plug-ins have completed their activities before continuing along the path after the stub. The selection policy for the example in Figure 3 could be *[Wednesday || Saturday]* for the first plug-in, *[any day]* for the second plug-in, and *[Saturday || Sunday]* for the third plug-in. New visual clues clearly become necessary with the introduction of new types of stubs. The *S* inside the stub indicates the synchronization.

Multiple Merge (Figure 3.c)—This pattern covers the situation where multiple branches are merged into one without synchronization but with the expectation that activities following the merge will be performed once for each active branch. The pattern is directly supported by an OR-join. Note that OR-constructs and AND-constructs in UCMs do not have to be properly nested and can therefore be used together.

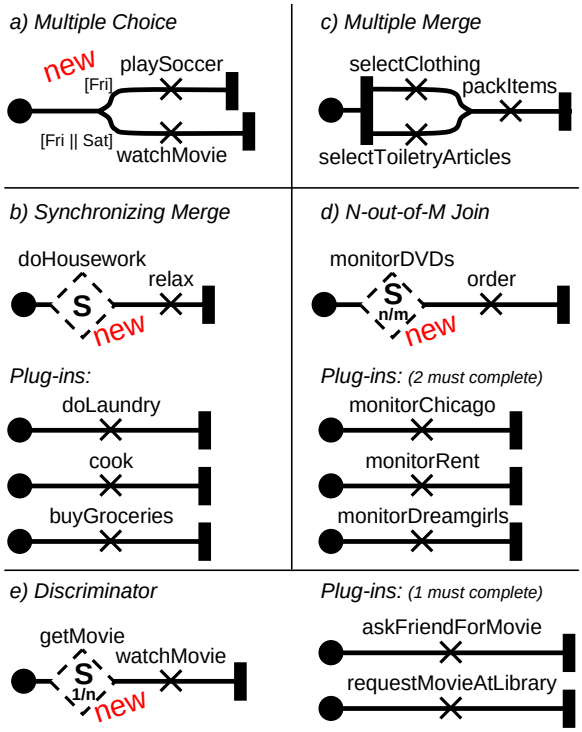

Figure 3: Advanced Branching and Synchronization Patterns (Group 2)

N-out-of-M Join (Figure 3.d)— This pattern is a type of merge with multiple concurrent incoming branches. The pattern is useful in a case where the n^{th} branch out of m incoming branches triggers the continuation of the workflow. All other incoming branches are ignored. Upon receipt of the last incoming branch, the n-out-of-m join is reset, so that the next set of incoming branches can once again trigger the continuation of the workflow. This pattern can be modeled with a more general version of the synchronizing stub for which N is specified. In this case, the selection policy of the stub enables all plug-ins. The traversal mechanism continues with the workflow once the n^{th} plug-in has completed. All other plug-ins do not trigger a continuation of the workflow. Once all plug-ins have completed, the stub is reset. $S_{n/m}$ inside the stub indicates the N-out-of-M Join pattern. The example in Figure 3.d describes the situation where a number of DVDs are monitored for availability and two DVDs are sufficient to receive free shipping (thus only two of the plug-ins need to complete before continuing).

Discriminator (Figure 3.e)—This pattern is a special case of the n-out-of-m join. It is a 1-out-of-m join and can be modeled similarly to the n-out-of-m join with a synchronizing stub. $S_{1/n}$ inside the stub indicates the Discriminator pattern.

3.4. Structural Patterns

Arbitrary Circles (Figure 4)—This pattern addresses non-structured cycles. The pattern is directly supported by the UCM notation since loops created with OR-forks and OR-joins do not have to be properly nested.

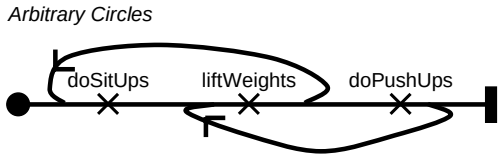

Figure 4: Structural Patterns (Group 3)

Implicit Termination—This pattern indicates that a workflow terminates automatically if there is nothing left to do. The pattern is directly supported by UCMs since there is no need to explicitly specify a termination responsibility in a UCM model. End points indicate the end of a workflow.

3.5. Patterns Involving Multiple Instances

Multiple Instances without synchronization (Figure 5.a)—This pattern describes the situation where an activity in a workflow needs to be executed multiple times in parallel without the need to synchronize any instances of the activity. This pattern can be modeled easily with a component and is therefore directly supported by UCMs. The UCM notation allows a *replication factor* to be specified for components, indicating whether one or more instances of a component take part in the scenario. The pattern, however, requires a specialized traversal mechanism capable of executing several instances of a component in parallel.

Multiple Instances with a priori known design time knowledge (Figure 5.b)—This pattern describes a point in the workflow where several instances of an activity have to be executed in parallel. These instances are synchronized in that the workflow continues only when all instances have been completed. The number of instances is known at design time. This pattern cannot be modeled with a replicated component since the instances of the component do not synchronize. The pattern, however, can be modeled concisely with a static synchronizing stub for which a replication factor is defined. The pattern is therefore directly supported by UCMs but requires a specialized traversal mechanism with the ability to define a replication factor for stubs. The plug-in of the stub is enabled the desired number of times in parallel, and the traversal mechanism waits until all plug-ins have been completed. In the example in Figure 5.b, S_{2x} inside the stub indicates the MI with a priori known design time knowledge pattern.

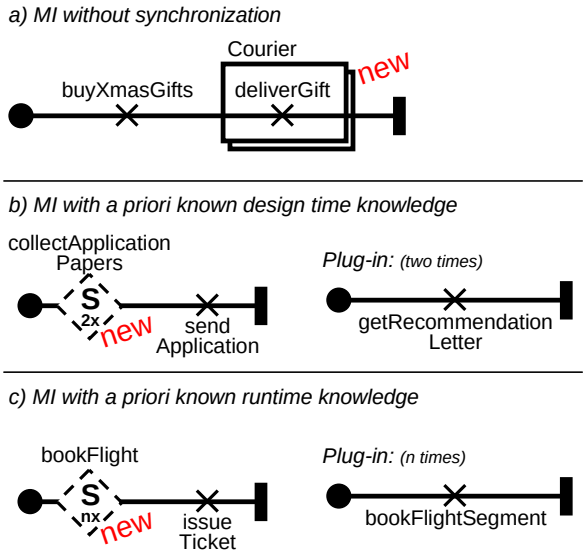

Figure 5: Patterns Involving Multiple Instances (Group 4)

Multiple Instances with a priori known runtime knowledge (Figure 5.c)—This pattern is the same as the previous one except that the number of instances is not known at design time but at runtime before the instances have to be created. Again, the pattern is directly supported by a static synchronizing stub. Its plug-in is enabled the desired number of times known at this point in the workflow, and the traversal mechanism waits until all plug-ins have been completed. Note that this requires scenario variables to be defined. S_{nx} inside the stub indicates the MI with a priori known runtime knowledge pattern.

Multiple Instances with no a priori runtime knowledge—This pattern is also very similar to the previous two patterns. The difference is that the number of instances is not known, not even while the instances are executing. The pattern could be modeled similarly to the two previous patterns but requires additional processing by the traversal mechanism in order to decide whether another plug-in is still required. This processing could be in the form of a condition that needs to be evaluated. The evaluation would have to take place while the already existing plug-ins are exe-

cuting. As this moves considerably further away from the original definition of a stub, this pattern is deemed to be not supported by UCMs. An example of this pattern is a court case with a callWitness activity. New witnesses may be called during the court case even when other witnesses already have been questioned. Only when all witnesses have been heard will the jury start deliberation.

3.6. State-Based Patterns

Deferred Choice (Figure 6.a)—This pattern describes a situation very similar to Exclusive Choice. The crucial difference is that the decision which branch to choose is not made at the choice point but implicitly by starting the first activity of one branch. In other words, all alternatives are possible until one alternative starts at which point all other alternatives are not available anymore (the other alternatives do not even start). An example for this pattern is the approval of a document which could be done either by the head of the department or by the project manager. The pattern can be modeled with UCMs with the help of an OR-fork (or a dynamic stub with plug-ins) without any specified conditions. The pattern is therefore directly supported but requires the traversal mechanism to understand such OR-forks or dynamic stubs.

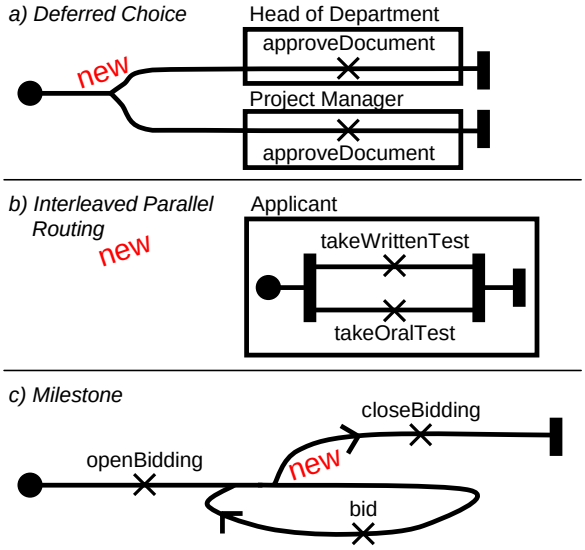

Figure 6: State-Based Patterns (Group 5)

Interleaved Parallel Routing (Figure 6.b)—This pattern describes the situation where two activities may be performed in any order but not in parallel. All examples given in [34] and [36] indicate a resource conflict as the reason for not being able to execute the activities in parallel. If this is the case, then UCMs with a specialized traversal mechanism directly support the pattern. The traversal mechanism can determine from the binding of responsibilities to components whether concurrent responsibilities are bound to the same component. If this is the case, then the concurrent branches are interpreted as Interleaved Parallel Routing. All other concurrent paths are interpreted as strictly parallel.

Milestone (Figure 6.c)—This pattern describes a point in the workflow where activity B can be executed (possibly multiple times) because activity A has already been executed and activity C has not yet been executed. The pattern is similar to Deferred Choice in that there is a race condition between two activities (B and C in this case) and only one activity is executed at one time. Milestone is different than Deferred Choice in that activity C is executed at some point. UCMs can model the Milestone pattern with an OR-fork without conditions and an implicit loop. Considering that direct support for this pattern in BPMN [33] is at least as complex as the UCM solution, this pattern is directly supported. The example in Figure 6 shows openBidding as activity A, bid as activity B, and closeBidding as activity C. A specialized traversal mechanism, however, is required for this pattern just as it is required for Deferred Choice.

3.7. Cancellation Patterns

Cancel Activity and **Cancel Case**—These patterns deal with the cancellation of an activity or workflow, respectively. The pattern can be used, among other things, to model exceptions. The UCM standard includes the *abort* construct which could be used for this purpose. jUCMNav, however, does not support this construct and the intended use of an abort makes it difficult to cancel activities or workflows which are not shown on the same UCM. Therefore, these patterns are deemed to be not supported by the UCM notation.

3.8. Communication Patterns

While UCMs abstract in general from the details of message exchange and communication infrastructures, it is still possible to model more detailed interactions between various architectural entities. The communication patterns represent different types of such interactions. The descriptions of the interaction type and the workflow descriptions, however, should not occur in the same UCM. UCMs describing the workflow should focus on the workflow alone, whereas other UCMs should focus on the interaction aspect. For instance, a *workflow UCM* may show two business part-

ners and a path which explains the services provided one after the other by each business partner. A path which crosses from the component of one business partner into the component of the other business partner represents some form of interaction. Multiple message exchanges may be necessary to achieve the interaction, but the workflow UCM abstracts away from that. The *interaction UCM* on the other hand shows the details of the interaction between these two components, allowing for a smoother transition into more detailed design. Interaction UCMs only have to be created once and are applicable to all instances of this interaction because the generic components in the interaction UCM can be bound to actual components in many workflow UCMs. Conceptually, one can think of the point in a workflow UCM where an interaction between components occurs as being replaced by a stub which contains one interaction UCM as a plug-in. Over the years, this concept has been discussed numerous times in the UCM community but tool support is not yet available for establishing links between workflow (or application) UCMs and interaction UCMs. Consequently, the identification of such a point in a UCM is also currently not supported, and therefore would have to be introduced as a new feature when the traversal mechanism is upgraded to support workflow description.

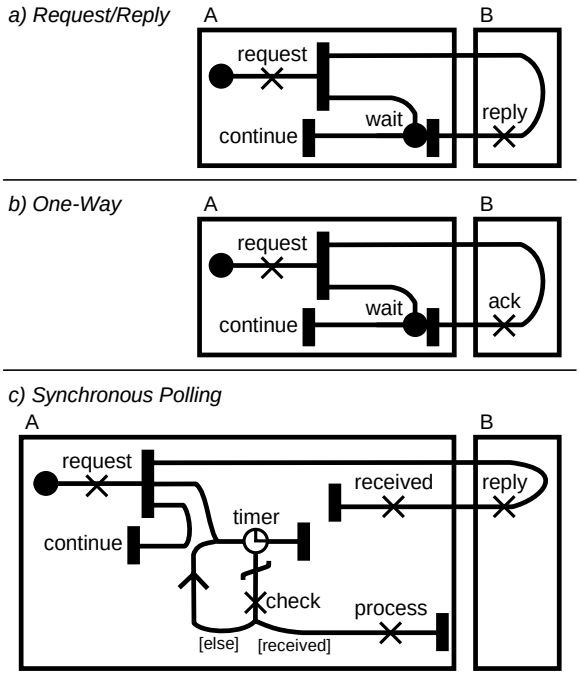

Figure 7: Synchronous Communication (Group 7)

The following paragraphs go through each of the six communication patterns (three synchronous and three asynchronous), showing how each of the patterns can be modeled by an interaction UCM.

Request/Reply (Figure 7.a)—This pattern describes a synchronous interaction where one side makes a request and then waits for a reply from the other side before continuing. This is modeled with a waiting place in UCMs.

One-Way (Figure 7.b)—This patterns is very similar to Request/Reply except that the reply is only an acknowledgement and does not contain any other information.

Synchronous Polling (Figure 7.c)—This pattern describes a synchronous interaction where one side makes a request but continues processing instead of waiting for a reply. At certain intervals, the requester checks whether a reply was received. If a reply was received, the reply is processed. If not, another timer is set for the next check. Note that the traversal mechanism requires a Boolean variable which is set to false at the *request* responsibility. The Boolean variable is evaluated at the OR-fork and set to true at the *received* responsibility.

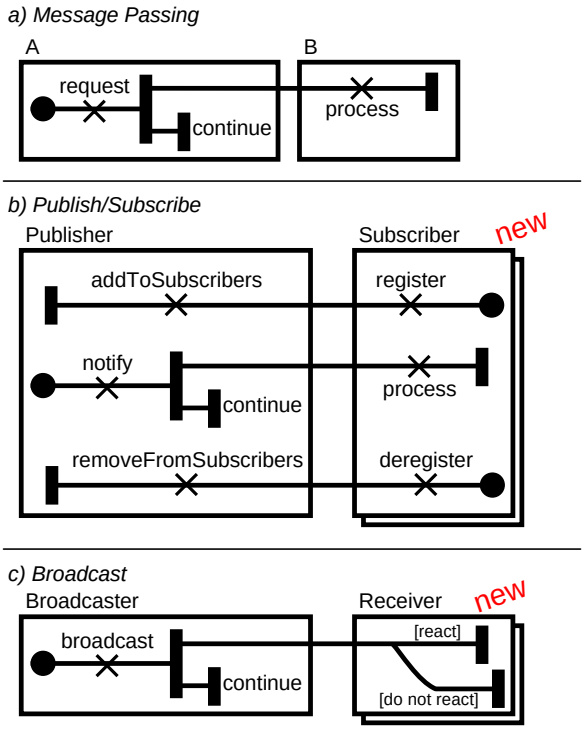

Figure 8: Asynchronous Communication (Group 8)

Message Passing (Figure 8.a)—This pattern describes an asynchronous interaction where one side makes a request and continues with the workflow. The other side processes the request.

Publish/Subscribe (Figure 8.b)—This pattern describes an asynchronous interaction where interested parties subscribe to a publisher and then process change notifications from the publisher until deregistering from the publisher. The interaction UCM makes use of the replication factor of components to indicate multiple subscribers. Therefore, the current traversal mechanism does not work for the pattern and has to be improved to properly deal with multiple instances of a component.

Broadcast (Figure 8.c)—This pattern describes an asynchronous interaction where a request is sent to all receivers in a network. Each receiver decides individually on how to react to the request. The interaction UCM makes use of the replication factor of components to indicate multiple receivers. Therefore, the current traversal mechanism does not work for the pattern and has to be improved to properly deal with multiple instances of a component.

3.9. Summary of Results

The results of the assessment are summarized in Table 1. BPMN 1.0 [33] directly supports all but one workflow pattern, UML 2.0 Activity Diagrams [33][35] directly support 17 out of 21, and BPEL4WS 1.1 directly supports 13 out of 21 workflow patterns [34]. UCMs support only eight of the workflow patterns if the current simple traversal mechanism is used. This increases to 18 with new notational elements and

Table 1 Comparison of UCMs using workflow and communication patterns

Type	Group	Pattern	UCMs	BPMN[a]	AD[a]	BPEL4WS[a]
Workflow Patterns	1	Sequence	Yes	Yes	Yes	Yes
	1	Parallel Split	Yes	Yes	Yes	Yes
	1	Synchronization	Yes	Yes	Yes	Yes
	1	Exclusive Choice	Yes	Yes	Yes	Yes
	1	Simple Merge	Yes	Yes	Yes	Yes
	2	Multiple Choice	Yes[b]	Yes	Yes	Yes
	2	Synchronizing Merge	Yes[b]	Yes	No	Yes
	2	Multiple Merge	Yes	Yes	Yes	No
	2	N-out-of-M Join	Yes[b]	Yes	Yes	No[c]
	2	Discriminator	Yes[b]	Yes	Yes	No
	3	Arbitrary Cycles	Yes	Yes	Yes	No
	3	Implicit Termination	Yes	Yes	Yes	Yes
	4	Multiple Instances without synchronization	Yes[b]	Yes	Yes	Yes
	4	Multiple Inst. with a priori known design time knowledge	Yes[b]	Yes	Yes	Yes
	4	Multiple Instances with a priori known runtime knowledge	Yes[b]	Yes	Yes	No
	4	Multiple Instances with no a priori runtime knowledge	No	No	No	No
	5	Deferred Choice	Yes[b]	Yes	Yes	Yes
	5	Interleaved Parallel Routing	Yes[b]	Yes	No	Partial
	5	Milestone	Yes[b]	Yes	No	No
	6	Cancel Activity	No	Yes	Yes	Yes
	6	Cancel Case	No	Yes	Yes	Yes
Communication Patterns	7	Request/Reply	Yes[d]			Yes
	7	One-Way	Yes[d]			Yes
	7	Synchronous Polling	Yes[d]			Yes
	8	Message Passing	Yes[d]			Yes
	8	Publish/Subscribe	Yes[bd]			No
	8	Broadcast	Yes[bd]			No

[a]results taken from [33] for the BPMN column, [33][35] for the AD column, and [34] for the BPEL4WS column
[b]assuming the existence of a tailored traversal mechanism (requires significant change to the current traversal mechanism).
[c]not assessed in [34], not supported because it is a general case of the Discriminator pattern.
[d]actually too detailed for the abstraction level of UCMs but can be modeled if desired.

a traversal mechanism specialized to workflow modeling. The results highlight that these improvements are essential to maintain the competitiveness of UCMs as a general scenario notation. In addition to these improvements, the support of cancellation patterns must be seriously considered for UCMs as all of them are supported directly by BPMN, Activity Diagrams, and BPEL4WS but not by UCMs. None of the compared languages directly supports multiple instances with no a priori runtime knowledge because only complicated workaround solutions exist. BPEL4WS 1.1 supports four out of six communication patterns [34]. UCMs support the same with the simple traversal mechanism and all with the specialized traversal mechanism. The desirable abstraction level of the UCM notation, however, is too high for the details of the six communication patterns. Therefore, the communication patterns should be out of scope and not shown on workflow-oriented or application-oriented UCMs, but, if desired, all of them can nevertheless be modeled on interaction-oriented UCMs.

Semantics is the biggest concern with regards to modeling scenarios or workflow with UCMs. UCMs can be interpreted in many different ways. This problem is addressed by the specialized traversal mechanism since the semantics of UCMs is more precisely defined by it. The specialized traversal mechanism introduces new kinds of stubs to the UCM notation: the synchronizing stub and a stub with multiple, concurrent invocations of its plug-in (either fixed or variable). Furthermore, the traversal mechanism now has to deal with OR-forks or stubs where several branch or plug-in conditions evaluate to true (see Multiple Choice) or no conditions are specified (see Deferred Choice). The traversal mechanism also has to identify Interleaved Parallel Routing based on the binding of responsibilities to components. In addition, the introduction of new types of stubs obviously requires new visual clues which were introduced in sections 3.3 and 3.5. See Table 2 for a list of requirements for the specialized traversal mechanism based on the findings of the preceding sections. The current traversal mechanism does not fulfill any of these requirements. The *UCM Status* column indicates whether the expected behavior of the traversal mechanism can be specified with the *current* UCM notation, with an *improved* notation as suggested in the preceding sections, or whether the specification is still an *open* issue.

In contrast to workflow patterns, communication patterns do not require a semantic clarification of the UCM notation as much as they require new organizational capabilities from a UCM tool. Application and interaction aspects have to be kept separate as their abstraction levels are different. A separate definition of application and interaction UCMs requires the identification of points in an application UCM where an inter-

Table 2 Requirements for evolving UCMs as a scenario and workflow description language

#	Requirement for Traversal Mechanism	Patterns (Source of Requirement)	UCM Status
1	Allow several branch conditions of OR-forks or several plug-in conditions of dynamic stubs to be true	Multiple Choice	Current
2	Allow OR-fork branches without any conditions or stub plug-ins without any selection policy	Deferred Choice, Milestone	Current
3	Support a synchronizing dynamic or static stub that specifies how many plug-ins have to complete	Synchronizing Merge, N-out-of-M Join, Discriminator, MI with a priori known design time/runtime knowledge[a]	Improved
4	Support a synchronizing static stub with variable or fixed number of parallel invocations of its plug-in	MI with a priori known design time/runtime knowledge	Improved
5	Support replication factor of components (multiple instances)	MI without synchronization, Publish / Subscribe Pattern, Broadcast Pattern, Communication Patterns	Current
6	Identify interleaving based on the binding of responsibilities to components	Interleaved Parallel Routing	Current
7	Support cancellation of activity or case (no suggestions given in this paper)	Cancel Activity, Cancel Case	Open
8	Support separate definition of workflow (application) UCMs and interaction UCMs	Communication Patterns	Current
9	Identify points in application UCMs linked to interaction UCMs	Communication Patterns	Open
10	Support bindings of components on plug-in maps to components on parent maps	Communication Patterns[b]	Open

[a]This is a special case of this pattern where only some of the parallel invocations have to complete before continuing.
[b]This is a general issue that should have already been resolved for the current traversal algorithm and is therefore not fully discussed in this document.

action occurs and links from such points to appropriate interaction UCMs.

The complexity of interaction UCMs is much greater than that of UCMs describing workflow patterns. The main reason for the complexity is that communication patterns, unlike workflow patterns, are not intended to be mapped to one modeling construct. Communication patterns result in interaction UCMs that represent a whole series of actions. Fortunately, the increased complexity is not a significant problem since interaction UCMs only have to be created once and are largely hidden from UCM designers.

4. Conclusion

This paper identified many semantic variation points in the UCM notation that require further formalization and suggested, as a possible solution, a specialized traversal mechanism and new notational elements. This mechanism defines the semantics of UCMs more precisely (i.e. in order to cover the patterns, the static UCM semantics does not change except for new categories of stubs, but the dynamic semantics needs further refinement in terms of how to deal with conditions and synchronization on branches and for stubs). Consequently, the improved UCM notation can model scenarios more uniformly across UCM applications.

UCMs are a general purpose scenario notation for describing behavior and structure, and therefore can certainly model the workflows described by BPMN and Activity Diagrams as well as the processes and activities used by BPEL4WS. This paper assessed the support of UCMs for workflow description in a structured and more formal way based on workflow and communication patterns. Although UCMs have been used for business process modeling, a formal assessment of the UCM notation has not yet been undertaken in this context. The results of the assessment were compared to the results of similar assessments for BPMN, Activity Diagrams, and BPEL4WS. The extent of the support is similar enough to make UCMs a good candidate for modeling workflow and scenarios in general if the UCM notation is improved as suggested. The greatest deficiency of UCMs is their lack of support for cancellation patterns. This highlights the need to evolve UCMs even further than suggested in this paper with the ability to model cancellations (and closely related exception handling) more efficiently.

Based on the assessment of UCMs, we presented a set of requirements for a specialized traversal mechanism in order to evolve the UCM notation as a general scenario language. These requirements can be used as a starting point for further discussion about appropriate traversal mechanisms. Future work will have to consider the high level of abstraction of UCMs. Are the proposed changes introducing too much detail into the UCM notation? Where is it necessary to draw a line in order to enable UCMs for workflow or web services description but still keep UCMs at a desirable high level of abstraction? When is it sufficient to have a workaround solution for a workflow pattern instead of direct support? Another avenue of research is the use of Aspect-oriented Use Case Maps (AoUCM) [18] for modeling of the communication patterns. Each communication pattern could be modeled as an aspect and composed with the rest of the UCM model according to composition rules. Such rules would define which components make use of which communication patterns. Furthermore, the UCM notation's capabilities should also be assessed from data and resource perspectives in order to gain a more complete overview of the applicability of UCMs for the description of workflows in particular and scenario descriptions in general.

Acknowledgement—This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Discovery Grants and Postgraduate Scholarships, and by the Ontario Research Network on e-Commerce. The author is grateful to Thomas Tran and Daniel Amyot for comments on drafts of this paper.

5. References

[1] Abdelaziz, T., Elammari, M., and Unland, R.: "Visualizing a Multiagent-Based Medical Diagnosis System Using a Methodology Based on Use Case Maps". *Multiagent System Technologies* (editors Lindemann-v. Trzebiatowski, G., Denzinger, J., Timm, I.J., and Unland, R.). LNCS 3187, Springer, September 2004, pp 198-212.

[2] Amyot, D. and Logrippo, L.: "Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System". *Computer Communication*. Vol. 23(12). July 2000, pp 1135-1157.

[3] Amyot, D., Charfi, L., Gorse, N., Gray, T., Logrippo, L., Sincennes, J., Stepien, B., and Ware, T.: "Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS". *Feature Interactions in Telecommunications and Software Systems VI*. Glasgow, Scotland, UK, May 2000, IOS Press, pp 274-289.

[4] Amyot, D., Roy, J.-F., and Weiss. M.: "UCM-Driven Testing of Web Applications". *SDL 2005: Model Driven* (editors Prinz A., Reed R., and Reed J.). LNCS 3530, Springer, June 2005, pp 247-264.

[5] Amyot, D., Weiss, M., and Logrippo L.: "UCM-Based Generation of Test Purposes". *Computer Networks*. Vol. 49(5), December 2005, pp 643-660.

[6] Amyot, D.: “Introduction to the User Requirements Notation: Learning by Example”. *Computer Networks*. Vol. 42(3), 21 June 2003, pp 285-301.
[7] Andrade, R.: “Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks”. *Lfm2000: Fifth NASA Langley Formal Methods Workshop*. Williamsburg, VA, USA, June 2000, pp 151-162.
[8] Billard, E.A.: “Operating system scenarios as Use Case Maps”. *ACM Workshop on Software and Performance*. 2004, pp. 266-277.
[9] Buhr, R.J.A. and Casselman, R.S.: *Use Case Maps for Object-Oriented Systems*. Prentice-Hall, 1996.
[10] Buhr, R.J.A.: “Use Case Maps as Architectural Entities for Complex Systems”. *IEEE Transactions on Software Engineering*. Vol. 24(12). December 1998, pp 1131-1155.
[11] Business Process Modeling Notation (BPMN) website; accessed April 2007: http://www.bpmn.org.
[12] Elammari, M. and Lalonde, W.: “An Agent-Oriented Methodology: High-Level View and Intermediate Models”. *1st International Workshop on Agent-Oriented Information Systems (AOIS)*. Heidelberg, Germany, June 1999.
[13] Gottschalk, K., Graham, S., Kreger, H., and Snell, J.: “Introduction to Web Services Architecture”. *IBM Systems Journal*. Vol. 41(2), 2002, pp 170-177.
[14] IBM WebSphere Software website; accessed April 2007: http://www.ibm.com/websphere.
[15] Roy, J.-F. Kealey, and Amyot, D.: “Towards Integrated Tool Support for the User Requirements Notation”. *SAM 2006: Language Profiles - Fifth Workshop on System Analysis and Modelling*. Kaiserslautern, Germany. LNCS 4320, pp. 198-215, Springer (2006); accessed April 2007: http://www.softwareengineering.ca/jucmnav.
[16] Kiepuszewski, B.: *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003.
[17] Microsoft BizTalk Server website; accessed April 2007: http://www.microsoft.com/biztalk/default.mspx.
[18] Mussbacher, G., Amyot, D., and Weiss M.: “Visualizing Early Aspects with Use Case Maps”. *To appear in Transactions on Aspect-Oriented Software Development*.
[19] OMG. UML 2.0 Superstructure Specification. OMG Formal Specification, formal/05-07-04, July 2005; acc. April 2007: http://www.omg.org/cgi-bin/doc?formal/05-07-04.
[20] Organization for the Advancement of Structured Information Standards (OASIS) Web Services Business Process Execution Language (WSBPEL) Technical Committee (TC); accessed April 2007: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
[21] Petriu, D.B. and Woodside, M.: “Software Performance Models from System Scenarios in Use Case Maps”. *Computer Performance Evaluation* (editors Field T., Harrison P.G., Bradley J., and Harder U.). LNCS 2324, Springer, April 2002, pp 141-158.
[22] Ruh, W.A., Maginnis,F.X., and Brown, W.J.: *Enterprise Application Integration – A Wiley Tech Brief*. John Wiley & Sons, Inc, 2001.
[23] Schofield, J.: “The third era starts here”. *Guardian Unlimited Technology*. May 29, 2003; accessed April 2007: http://technology.guardian.co.uk/online/story/0,3605,965532,00.html.
[24] Siddiqui, K.H. and Woodside, C.M.: “Performance aware software development (PASD) using resource demand budgets”. *Workshop on Software and Performance (WOSP)*. Rome, Italy, July 2002, pp 275-285.
[25] Simple Object Access Protocol website; accessed April 2007: http://www.w3.org/TR/soap.
[26] Universal Description, Discovery, and Integration website; accessed April 2007: http://www.uddi.org.
[27] *URN - Goal-oriented Requirement Language (GRL), ITU-T Draft Recommendation Z.151*. Geneva, Switzerland, Sep. 2003; acc. April ‘07: http://www.UseCaseMaps.org/urn.
[28] *URN - Use Case Map Notation (UCM), ITU-T Draft Recommendation Z.152*. Geneva, Switzerland, Sep. 2003; accessed April 2007: http://www.UseCaseMaps.org/urn.
[29] *User Requirements Notation (URN) – Language Requirements and Framework, ITU-T Recommendation Z.150*. Geneva, Switzerland, February 2003; acc. April 2007: http://www.itu.int/ITU-T/publications/recs.html.
[30] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P.: “Workflow Patterns”. *Distributed and Parallel Databases*. Vol 14(3), July 2003, pages 5-51.
[31] Web Services Description Language website; accessed April 2007: http://www.w3.org/TR/wsdl.
[32] Weiss, M. and Amyot, D.: “Business Process Modeling with URN”. *International Journal of E-Business Research*. Vol. 1(3), July-September 2005, pp 63-90.
[33] White, S.A.: “Process Modeling Notations and Workflow Patterns”. *Workflow Handbook 2004* (editor: Fischer, L.). Future Strategies Inc., 2004, pp 265–294; acc. April ‘07: http://www.bpmn.org/Documents/Notations and Workflow Patterns.pdf.
[34] Wohed, P., van der Aalst, W.M.P., Dumas, M. and ter Hofstede, A.H.M.: “Analysis of Web Services Composition Languages: The Case of BPEL4WS”. *Proceedings 22nd Intl. Conference on Conceptual Modelling (ER)*. Chicago IL, USA, October 13-16, 2003, pp. 200-215.
[35] Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., and Russell, N.: “Pattern-based Analysis of UML Activity Diagrams”. *BETA Working Paper Series*. WP 129, Eindhoven University of Technology, Eindhoven, Netherlands, 2004; accessed April 2007:
http://is.tm.tue.nl/research/patterns/download/uml2patterns BETA TR.pdf.
[36] Workflow Patterns website; accessed April 2007: http://www.workflowpatterns.com.
[37] YAWL: Yet Another Workflow Language website; accessed April 2007: http://yawlfoundation.org/.