Workflow Wrapper for Unstructured Workflows

Boris Milašinović, Krešimir Fertalj

Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia

boris.milasinovic@fer.hr

kresimir.fertalj@fer.hr

Abstract— **Modelling workflow for prerequisite relations produces well-defined and well-formed workflow that contains only and-splits and and-joins. Such workflow often cannot be transformed to a structured workflow. Workaround based on elements cloning produces similar structured workflow that corresponds to original workflow, but requests design of a workflow wrapper that does necessary adjustments in the runtime. This paper presents key elements of an implementation of such workflow wrapper using Windows Workflow Foundation and describes the data structure for storing data about clones and methods to properly handle clones in the runtime. Moreover, presented solution use web services and as such extends the number of supported client platforms.**

Keywords— **workflow management, workflow wrapper, unstructured workflows. Windows Workflow Foundation**

I. Introduction

Decoupling the system components is essential to keep a system flexible enough to handle unnecessary changes during development and maintenance. It usually means using multi layer architecture where at least three layers should exist: presentation layer, business layer and data layer. As noted in the guidelines for designing the business layer of an application [12] one of the tasks would be to separate workflow component inside business layer. Recognizing business process, implementing it successfully and enabling further changes are crucial for automating company's operations and here lies the importance of the workflow component. Workflow separates when and in which order to do from how it should be done and ease maintenance of the system. Moreover, graphical representation of the workflow makes business process easier to understand to wider spectrum of people.

A. *Workflow Appliance and Diversity*

Although traditionally related to enterprise systems, workflow management finds its usage in various types of applications, especially in those where some prerequisite relations has to be modelled. Some appliance of prerequisite relations between workflow elements can be found in merging dependencies between UML components [6], in modelling course prerequisites in learning management systems [3], [10], [16], in modelling relationships between workflow sub-components of a learning management system [22] or in modelling workflow-based data integration for e-learning systems [21].

Decision to use workflow component is just one piece of business process modelling puzzle. Choosing appropriate workflow management software is not an easy task. As noted in [1] in year 2000 existed more than 30 proprietary workflow management systems and the number increased during the years with the numerous open source systems. Despite WfMC's [20] efforts to standardize the systems and ensure interoperability, different approaches and company policies produced several standards. In [8] standard workflow patterns had been defined and survey has been done that shown that most of the systems support just basic workflow patterns. Further problem is that even those basic patterns are not enough to model simple prerequisite relations as it will be shown in the next sections.

For sake of presentation in this article authors decided to use Windows Workflow Foundation [19] (in further text WF) as it is integral part of .NET framework, it is interoperable with other systems using web services and it supports persistence. When dealing with workflows it is assumed that workflow component defines and coordinates long running, multistep business processes. Persistence service is needed to save the state of the idled workflow and to restore it again when interaction with workflow is needed. In this way, memory resources are saved and data loss is avoided in case of host restart. Moreover, using WF it is possible to start a workflow from one type of application (e.g. web application) and continue it from another type of application (e.g. windows application).

B. *Usage Restrictions*

While using workflows the authors encountered two main problems that have to be solved – expanding use of workflows in heterogeneous environment (especially in those including mobile applications) and modelling unstructured workflows. Formal definition of structured workflow is given in [7] where is stated that many workflow management systems allow only structured workflows. As it will be shown in the next chapter this limitation can be significant obstacle when modelling prerequisite relationships.

Both problems are further elaborated in the next chapter, followed by the proposed solution in the third chapter. The proposed solution consists of writing a workflow wrapper that will either act as a proxy to applications that will otherwise be unable to use created workflows and also as a separate layer that handles complexity of the proposed workaround for unstructured workflows.

II. Main Problems

A. Limitation based on client platform

As system can consist of heterogeneous clients and components, defining the layer boundaries and successfully communicating between them is necessary to have the functional system. In such scenario exposing workflows and content delivery as a web service is essential to extend the scope of usage. Similar principle was used in some e-learning systems [18], [4]. In this way, better separation of layers is enabled and set of possible client's platforms has been extended [17]. Workflows created using WF and exposed as Windows Communication Foundation (WCF) service are theoretically available to all types of clients that can use SOAP protocol to consume a web service. Practical problem occurs with long running workflows where persistence service has to be used. For such situations clients must support context binding that creates context channel for communication between client and workflow service. Context channel manages the unique conversation identifiers for the interactions. Conversation identifier enables recognition of a particular workflow element that needs to receive the particular message as shown in Fig. 1. The problem with the context binding is that it is not supported on all types of client platforms. An example of a client that cannot use such context binding directly is a mobile application. A workaround consists of creating a new web service that will act as a proxy for those clients. This is done as described in chapter III as a part of a proposed workflow wrapper.

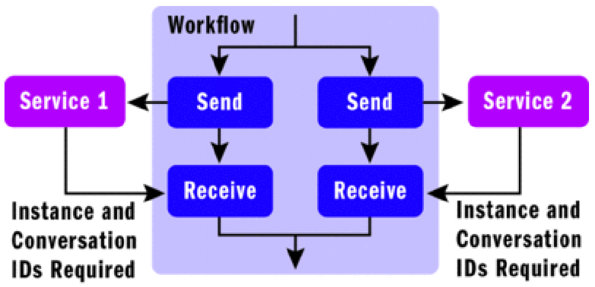
Workflow
Service 1
Send
Send
Service 2
Receive
Receive
Instance and Conversation IDs Required
Instance and Conversation IDs Required

Fig. 1 Conversation management using workflow instance and conversation identifier [14]

B. Unstructured workflows

Prerequisite relations between elements can be graphically presented with directed arcs between vertices. Direction of an arc defines dependency of target vertex on source vertex. If the graph is connected and have only one source and one sink vertex, graph can be mapped to well-formed workflow but not necessary to structured workflow. In [7] authors define workflow as a well-behaved if it can never lead to deadlock nor it can result in multiple active instances of the same activity. Another research [5] defines well-formed workflows, relate them to well-behaved workflows and define prerequisites that well-formed workflow must have in order to be structured. Tools for transforming a process model into a corresponding structured model [15] and for translating an unstructured workflow to a structured BPEL model [9], [2] automate the transformation, but not all models can have their structural pair. Moreover, in [11] it has been shown that workflow containing only and-splits [20] is always well-behaved but does not have a structured mapping if it does not meet certain requirements. Those requirements are, intuitively expressing, that each and-split is paired with corresponding and-join. In [13] it is shown that already simple prerequisite modelling produces graphs that cannot have structured mapping and provided workaround for that. Workaround consists of element cloning (duplication) with adjustment in the runtime that would ensure that clones behave as single element and that no data duplication occurs.

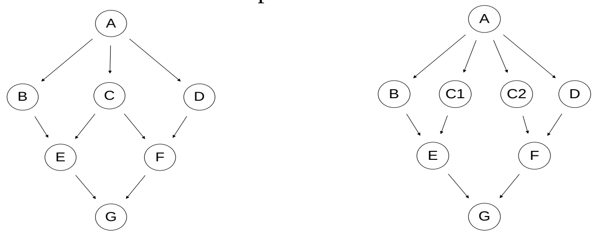

Fig. 2 Unstructured workflow and resulting structured workflow

Before describing how these runtime adjustments should be implemented, the idea from [13] is briefly described using two graphs from Fig. 2. The left graph corresponds to an unstructured workflow. If vertex C is cloned into two instances C1 and C2, and if C1 is prerequisite for E and C2 is prerequisite for F that would lead to structured workflow model presented by the right graph. The right graph is equivalent to a structured workflow that can easily be transformed into a WF-model (or any other concrete workflow model). In the runtime it has to be assured that C1 and C2 are shown as a single instance of C. [13] has described how to find elements that have to be cloned but did not specify how runtime adjustments have to be done. Although clone handling can be done anywhere, writing workflow wrapper hides duplication problems from other layers in the system and exposes only unique instances to other layers of the application.

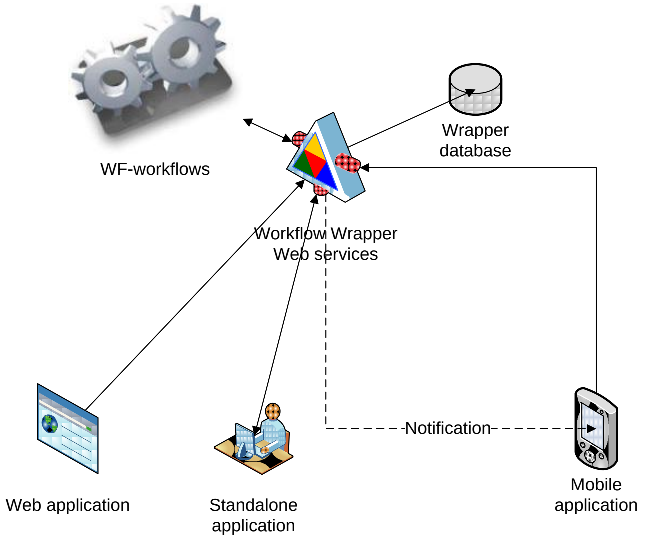

Fig. 3 Architecture diagram of a workflow wrapper and involved components

III. Workflow Wrapper

Fig. 3 shows the architecture diagram that cover scenarios in which workflow wrapper can be used. Workflow wrapper will act as a proxy for clients that cannot support WF directly and it will encapsulate complexity of proper handling of element clones. Specially adapted content delivery and notifications about newly started elements for clients that are not always connected (such as mobile clients) are done by writing custom plugins for a particular client. Data about clients, running workflows and plugins are stored in wrapper configuration database. (Note: Development of those plugins is out of the scope of this paper.)

A. Wrapper Internal Structure

Internal structure of the workflow wrapper consists of several configuration files that define WCF binding configurations and the database that contains four tables shown in Fig. 4. Purpose of the database is to store and retrieve data about elements and workflows. When new instance of a workflow is started, workflow instance identifier is assigned to user and saved to *WorkflowInstance* table. Context of the workflow instance is saved to *WorkflowInstanceContext*. When new element is started workflow instance identifier and element identifier (in a particular workflow) are paired with newly created element’s global unique identifier and its conversation identifier is stored in *ConversationId* table. Conversation identifier of any later started clone of an existing element is also stored in the *ConversationId* table, but new global unique identifier for that element is not created and inserted in *Element* table. Data in tables *WorkflowInstanceContext* and *ConversationId* are used to reconstruct channel context and deliver message to appropriate element in the workflow.

Communication between system components is regulated with WCF communication contract shown in Table I. Workflow wrapper implements all of these methods, but of a particular interest are two methods: *ElementStarted* and *ElementFinished*.

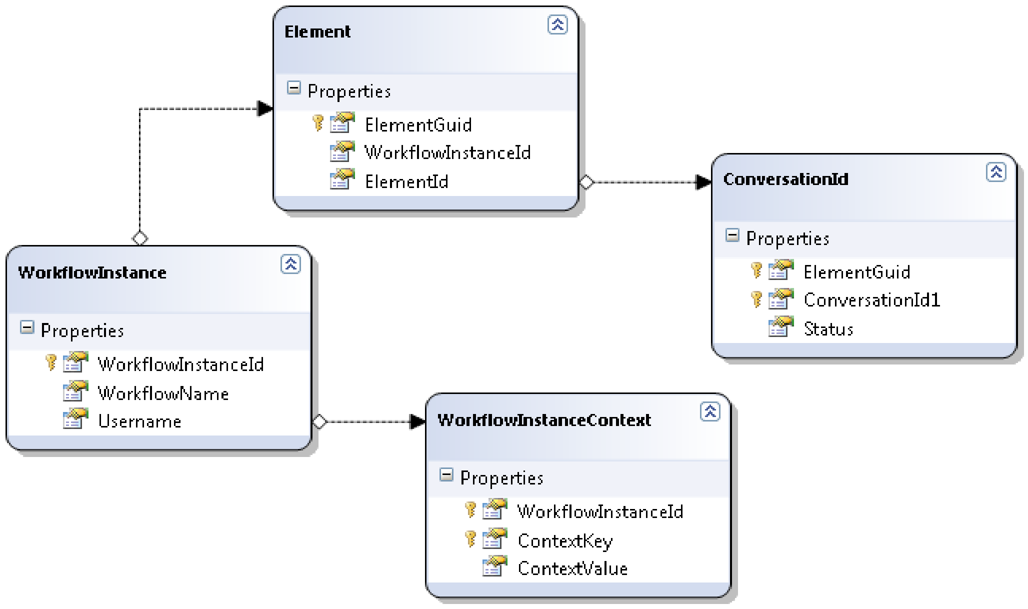

Fig. 4 Workflow wrapper database

TABLE I
COMMUNICATION CONTRACT METHODS

```
public interface IWFCommunicationContract{
        [OperationContract] Dictionary<string, string> GetMyWorkflows(string username);
        [OperationContract] string StartWorkflow(string WorkflowName);
        [OperationContract] string CancelWorkflow(string WorkflowName);
        [OperationContract] void ElementStarted(string ElementId, string WorkflowInstanceId,
                                                string ConversationId);
        [OperationContract] void ElementFinished(string ElementGuid);
        [OperationContract] void CancelElement(string ElementId);
        [OperationContract] void TimeElapsed(string ElementId);
        [OperationContract] List<string> GetWFStatus(string WorkflowInstanceId);
        [OperationContract] byte[] GetWFStatusImage(string WorkflowInstanceId);
}
```

B. Single element conversion

Each element from the prerequisite model can be represented with two WF-activities as shown in Fig. 5. (Note: Figure has been simplified for the sake of presentation. Concrete implementation may also contain additional activities if element cannot be started until some specific time or must be finished before some specific time)

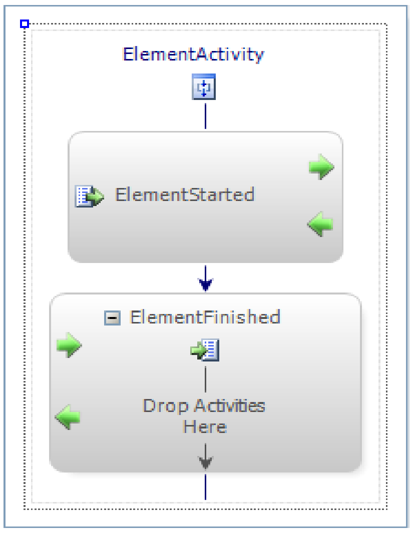

Fig. 5 Custom WF-activity for each element

The first activity is of type *SendActivity* bound to method *ElementStarted* from the communication contract. The purpose of this activity is to notify external system (in this case workflow wrapper) that particular element is activated. As it does not carry any specific semantic meaning it is on the external system to interpret what it means and undertake further actions as running plugins for notifications and/or content delivery. When an element is finished, client will call method *ElementFinished* on workflow wrapper web service. After that, workflow wrapper will call the same method on all element clones and that calls will activate the second activity from Fig. 5 – activity of type *ReceiveActivity* bound to method *ElementFinished*. As all elements have these two activities, recognizing correct *ReceiveActivitiy* that should be activated on incoming request is determined using the unique conversation identifiers.

C. Workflow wrapper's main algorithms

In addition to extending number of supported platforms, workflow wrapper purpose is to hide complexity of handling elements clones. This is done by implementing methods *ElementStarted* (of type *SendActivity*) and *ElementFinished* (of type *ReceiveActivity*). Pseudo code for the implementation of these two methods in the workflow wrapper is shown in Table II. When custom WF-activity from Fig. 5 starts, it calls *ElementStarted* method on the workflow wrapper and sends element identifier, current workflow instance identifier and conversation identifier of *ReceiveActivity* that follows immediately after itself. Workflow wrapper then looks up in table *ElementId* searching if this element has been already started which would mean that it is a clone of an existing element. If started element is not a clone, workflow wrapper creates new global unique identifier for this element and loads notification and action module (contained in the appropriate plugins). In either way, it saves received conversation identifier as it will be used in *ElementFinished* method.

When *ElementFinished* method is called, workflow wrapper loads data stored for received element global identifier. Data contains workflow instance identifier, workflow instance context and data about all clones of the element bound to this global identifier. For each conversation identifier, wrapper will call corresponding *ReceiveActivity* in the workflow instance and in this way, all clones will be finished.

TABLE II
PSEUDO CODE OF *ELEMENTFINISHED* AND *ELEMENTSTARTED* IMPLEMENTATION

```
ElementStarted(elementId, wfInstanceId, conversationId)
        elementGuid := GetFirstGuid(elementId, wfInstanceId)
        if elementGuid is not defined then
                elementGuid := InsertElement(elementid, wfInstanceId)
                InvokeNotificationModule(elementId, wfInstanceId, elementGuid)
                InvokeActionModule(elementId, wfInstanceId, elementGuid)
        SaveConversationId(elementGuid, conversationId)


ElementFinished(elementGuid)
        wfInstanceId := GetWorkflowInstanceId(elementGuid)
        context := LoadContextData(wfInstanceId)
        workflowName := GetWorkflowName(wfInstanceId)
        binding := GetBindingTypeAndEndpoint(workflowName)
        channelFactory := CreateWCFChannelFactory(binding)
        C := GetConversationIds(binding)
        for each conversationId in C do
                NotifyWorkflow(context, conversationId, channelFactory)
```

IV. Conclusion

Modelling workflow for prerequisite relations produces well-defined and well-formed workflow that often cannot be transformed to a structured workflow. Due to this, either some modifications to the model has to be introduced or custom workflow management software has to be written. Solution with cloning elements and designing workflow wrapper for handling clones represents a compromise in order to use existing workflow management software with minimal development of custom software. Moreover, as most of the workflow management software limit the number of the clients based on clients platform, presented workflow wrapper that use web services for communication extends the number of supported client platforms. Extensibility of a system is achieved using two types of plugins: plugin that will notify disconnected users that new element has been started, and plugin for invoking concrete action (content delivery or even start of a new workflow) for the started element. In this way workflow just models relationship between elements without further tight coupling with element semantic.

References

[1] W. van der Aalst and K. van Hee, *Workflow management. Models, Methods and Systems*, MIT Press, Cambridge, Massachusetts, USA, 2004.

[2] W. van der Aalst, and K. Lassen, “Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation,” *Information and Software Technology*, vol. 50, no. 3, pp 131-159, 2008

[3] M. Cesarini, M. Monga, and R. Tedesco, “Carrying on the e-Learning process with a Workflow Management Engine,” in *Proc. ACM Symp. Applied Computing*, Nicosia, Cyprus, 2004, pp 940-945.

[4] K. Fertalj, N. Hoić-Božić, and H. Jerković, “The Integration of Learning Object Repositories and Learning Management Systems,” *Computer Science and Information Systems*, vol. 7, no. 3, pp 387-407, 2010.

[5] R. Hausser, M. Friess, J. M. Küster, and J. Vanhatalo, “Combining Analysis of Unstructured Workflows with Transformation to Structured Workflows,” in *Proc. 10th IEEE Int. Enterprise Distributed Object Computing Conference*, Hong Kong, China, 2006, pp 129-140.

[6] Y. B. Hlaoui, and L. J. B. Ayed, “An Interactive Composition of UML-AD for the Modelling of Workflow Applications,” *Ubiquitous Computing and Communication Journal*, vol. 4, no. 3, pp. 599-608, 2009.

[7] B. Kiepuszewski, A ter Hofstede, and C. Bussler, “On Structured Workflow Modelling,” *Lecture Notes in Computer Science*, vol. 1789, pp. 431-445, 2000.

[8] B. Kiepuszewski, “Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows,” Ph.D. dissertation, Queensland University of Technology, Brisbane, Australia, 2003.

[9] K. Lassen, and W. van der Aalst, “WorkflowNet2BPEL4WS: A Tool for Translating Unstructured Workflow Processes to Readable BPEL,” *Lecture Notes in Computer Science*, vol. 4275, pp. 127-144, 2006.

[10] L. Lin, C. Ho, W. Sadiq, and M. E. Orlowska, “Using Workflow Technology to Manage Flexible e-Learning Services,” *Educational Technology & Society*, vol. 5, no. 4, pp. 116-123, 2002.

[11] R. Liu, and A. Kumar, “An analysis and taxonomy of unstructured workflows,” *Lecture Notes in Computer Science*, vol. 3649, pp. 268-284, 2005.

[12] Microsoft Patterns & Practices Team, *Microsoft Application Architecture Guide (Patterns and Practices)*, 2nd ed., Microsoft Press, Redmond, USA, 2009.

[13] B. Milašinović, and K. Fertalj, “Using partially defined workflows for course modelling in a learning management system,” *The 4th Int. Conf. Information Technology (ICIT-2009)*, Amman, Jordan, 2009.

[14] M. Milner. (2008) Workflow Services. MSDN Magazine, Launch Issue [Online] Available: http://msdn.microsoft.com/en-us/magazine/cc164251.aspx#S5

[15] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, “Structuring Acyclic Process Models,” *Lecture Notes in Computer Science*, vol. 6336, pp. 276-293, 2010.

[16] S. Sadiq, W. Sadiq, and M. Orlowska, “Workflow Driven e-Learning – Beyond Collaborative Environments,” *International NAISO Congress on Networked Learning in a Global Environment, Challenges and Solutions for Virtual Education*. Berlin, Germany, 2002.

[17] T. Vantroys, and J. Rouillard, “Workflow and Mobile Devices in Open Distance Learning,” in *Proc. IEEE Int. Conf. Advanced Learning Technologies*, Kazan, Russia, 2002, pp. 123-127.

[18] G. Vossen, and P. Westerkamp, “E-learning as a Web service (extended abstract)”, in *Proc. 7th Int. Conf. Database Engineering and Applications*, Hong Kong, China, 2003, pp. 242-249.

[19] (2010) Windows Workflow Foundation website. [Online]. Available: http://msdn.microsoft.com/en-us/netframework/aa663328.aspx

[20] (2010) Workflow Management Coalition website. Terminology & Glossary: Document Number WFMC-TC-1011 - Document Status - Issue 3.0 [Online]. Available: http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html

[21] S. Xi, and J. Yong, “New Data Integration Workflow Design for e-Learning,” *Lecture Notes in Computer Science*, vol. 4402, pp 699-707, 2007.

[22] J. Yong, “Workflow-based e-learning platform,” in *Proc. 9th Int. Conf. Computer Supported Cooperative Work in Design*, Coventry, UK, 2005, vol. 2, pp. 1002-1007.