Workflow Charts and Their Semantics Using Abstract State Machines

Theodorich Kopetzky, Verena Geist

Software Competence Center Hagenberg GmbH
Softwarepark 21, 4232 Hagenberg, Austria
{theodorich.kopetzky,verena.geist}@scch.at

Abstract: Workflow charts are a novel way to describe business processes and a way of putting more emphasis on the human-computer interaction. They introduce a typed approach to workflow specification in order to improve flexibility in business process technology by proposing a concept for integrating workflow definition and dialogue programming, being also open to business process modelling. Although their precursor has its semantic specification based on Unified Modelling Language semantics, workflow charts currently lack a complete formal semantic specification. Instead of enhancing the Unified Modelling Language semantics, the approach of specifying the semantics of workflow charts using the formalism of Abstract State Machines has been chosen. The resulting description provides a precise operational semantics.

1 Introduction

Business process-related topics are an active field of research, one subset being the different languages to describe business processes. While many languages are more focused on the control flow of the business process, e.g. Yet Another Workflow Language (YAWL) [vdAtH05] or the Business Process Model and Notation (BPMN) [OMG11], there are languages which focus on different aspects, for example subject-oriented process management [FSS+11], where the focus is more on the subjects and their tasks. Modeling of the user interaction is usually not in the focus of those languages.

While workflow charts [Dra10] still model the control flow they put more emphasis on user interaction by implementing a submit/response-style user interaction pattern. The interaction with a submit/response-style system consists in a continuous interchange of report presentations and form submissions. This user interface paradigm is widely used in form-based applications, ranging from simple web applications to complex ERP systems. The concept of workflow charts relies on a methodology to model form-based applications called formcharts [DW04]. However, they lack a complete formal semantic specification. To meet the needs of business processes, formcharts are extended to workflow charts in due consideration of the worklist paradigm, concurrency, and actors as well as roles. A formalisation of workflow charts will provide a precise operational semantics. Thus, workflow charts represent a technology-independent, conceptual modelling language and are at

the same time designed for integrating workflow definitions and dialogue programming to create an executable specification language, providing a more flexible process technology [Gei11]. The reason for creating such formalisation is twofold. On the one side, we want to be able to reason about workflow charts and simulate models without writing a prototype in a conventional programming language. The implementation with CoreASM (see Section 4 on page 11) enables us to do that in a short and concise way. On the other side, seeing how languages as BPMN lack a precise formal specification and the problems that may lead to (e.g., contradictions in the BPMN-specification in [Nat11]), or Unified Modelling Language (UML) which has unclarities in its semantics [FSKdR05], such a specification seems the prudent way to avoid those problems.

In this paper we formalise the concept of workflow charts using Abstract State Machines (ASMs). In Section 2, we first discuss the nature of workflow charts, using a descriptive example. In Section 3.1 we give a very short introduction to ASMs. In Section 3, we discuss the semantics of workflow charts using ASMs including descriptions of the assumed environment, required nodes and associations, and miscellaneous static functions. We also provide the definition of the ASM agent that operates on a given workflow chart. A brief overview of the implementation of the specified semantics using CoreASM is given in Section 4. In Section 5, we provide a short survey of related work. We sum our main findings up in Section 6 and also comment on the future work.

2 Workflow Charts

Workflow charts as a modelling language for business processes have been introduced in [Dra10]. They are specified as a tripartite graph, follow a submit/response-style user interaction pattern, and consist of the following node types:

Client Pages (Client Pages (CPs)) represent computer screens that show information and provide forms for user input. They are specified using ellipses in workflow charts.

Immediate Server Actions (ISAs) represent forms that appear on computer screens and are specified using rectangles.

Deferred Server Actions (DSAs) appear as links in the worklists of the corresponding actors and are specified as rectangles with a gradient fill.

The edges, or associations, have a twofold meaning: On the one hand, they communicate the flow of control through the graph and, on the other hand, they function as associations defining which data is to be presented to an actor. The associations are usually guarded by differently named kinds of conditions, which correspond to guards in the ASM ground model. A very simple workflow charts graph can be seen in Fig. 1 on the next page.

One of the fundamental metaphors of workflow charts is the worklist metaphor. The worklist is the one point where actors in a workflow[1] choose the next step in the workflow to

[1] We use the notion of a workflow as an executable business process specification synonymously to the notion of a business process. This seems more natural here as this word is part of the name "workflow charts", too.

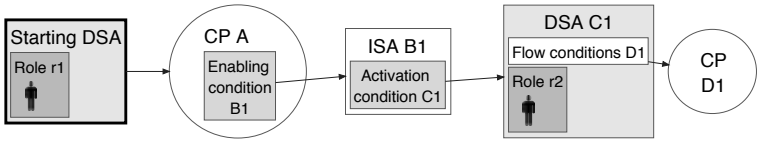

Figure 1: A very simple workflow chart.

execute. Each actor has his own worklist resp. his own view of the global worklist. The worklist is divided into two areas, i.e., the workflow area and the task area. The workflow area (start menu) contains all registered workflows that can be started by the actor. The entries in the task area are related to DSAs in a way that each DSA available for execution has a representation in the worklist. Starting a workflow leads to the appearance of a new entry, represented by the starting DSA, in the task area. If the worklist for all actors is empty, the workflow as a whole has finished. In addition, a dialogue processor and a worklist processor are assumed as parts of a hypothetical workflow system which executes the workflow chart specification. The details of this system can be looked up in [Gei11].

CPs and ISAs are both typed. Both node types represent the basic user interaction of a workflow. CPs specify through their type and their reporting action which data is displayed to the actor, whereas ISAs specify through their type the data an actor can possibly enter to modify the system state on submission. CPs and ISAs are associated via so called enabling conditions. If an enabling condition evaluates to *true*, the respective data input possibility according to the associated ISA is presented to the actor. The representation and evaluating of conditions is managed by the aforementioned dialogue processor. It is a characteristic of workflow charts that the CPs and the ISAs are modelled as nodes but that they are not visited separately during execution of the workflow. Instead one CP and its corresponding ISAs are handled as a kind of a meta-node.

If the actor enters and submits data, he can only submit data for one ISA, even in the presence of multiple ISAs to choose from. The workflow system evaluates the different activation/show conditions (we call them activation conditions from now on) which are associated to the activated ISA. One ISA can have $0, \ldots, n$ activation conditions, and they may all evaluate to *true*. For each activation condition there is an associated DSA. If m activation conditions of an ISA evaluate to *true*, m corresponding DSAs are activated, resp. inserted into the actor's worklist.

Active DSAs are represented as workflow items in the worklist which an actor can choose from. Typically the worklist processor shows these items as links to the actor and thus the type of a DSA is void. If the actor chooses a workflow item, the corresponding DSA activates exactly one CP from the available n CPs. This is determined by the flow conditions, which are specified with the DSA. Only one of them may evaluate to *true*. The determined CP in combination with the associated enabled ISAs is then presented to the actor.

Fig. 2 on the following page shows the different elements of workflow charts in one diagram. In addition to the already specified nodes, links, and conditions one can also see roles. Roles are used to specify which actor has access to which worklist item (DSA). The

different conditions (enabling conditions, activation conditions, and flow conditions) are evaluated against the global system state, which is not specified in detail here.

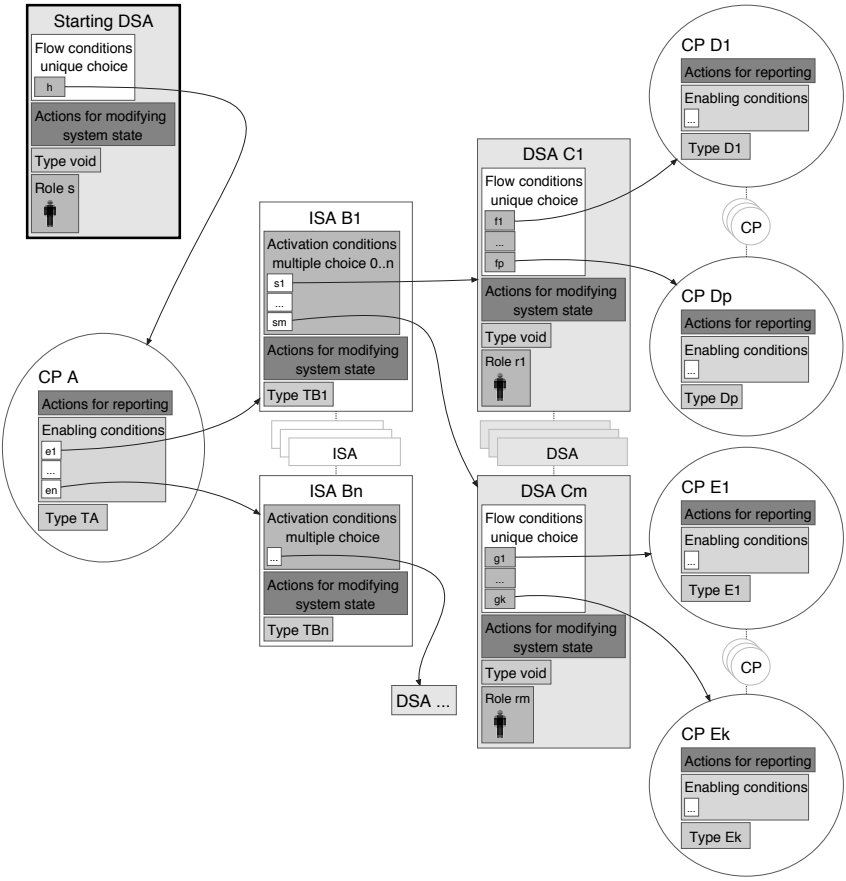

Figure 2: Complete workflow chart at a glance.

Example of Workflow Charts

To stress the understanding of workflow charts we present a small example, the beginning of a Business Trip Application Process (BTAP)[2] in Fig. 3 on the next page. The given example presents a business trip (trip from now on) workflow that deals with the tasks of trip application, review, approval and cancellation. The workflow is the scientific outcome of a comprehensive case study in association with the Austrian Social Insurance Company

[2]We are aware of the many different styles of BTAPs and that this model falls short in covering all of them.

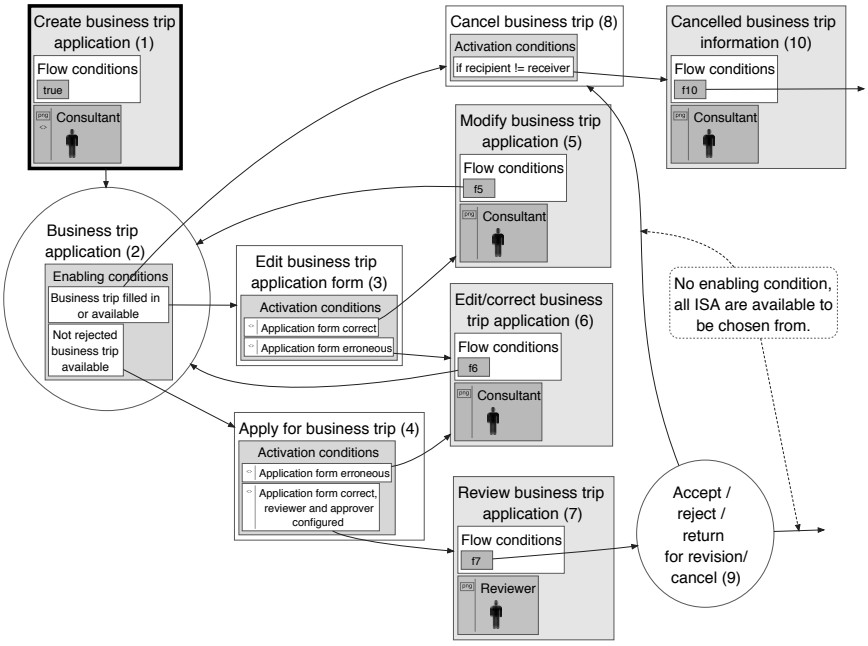

Figure 3: Beginning of a BTAP using workflow charts.

for Occupational Risks (AUVA) [DN08]. The different nodes of the workflow chart in the figure are numbered in parenthesis after the name of the node.

For the example we assume that the actor, in the figure named “consultant”, can select items from his worklist and that he uses a user interface, which allows in- and output of data and which is serviced by a system that is unspecified otherwise. In addition, there is a second actor in the diagram, “reviewer”, whom we will need at the end of the example.

Additionally, we assume that the consultant starts with no pre-existing trip applications. Then possible actions in this model are as follows:

- The consultant starts the workflow by selecting a representation of node (1) of Fig. 3 from his worklist.
- The system presents the information specified in node (2), a CP, to the consultant. In addition, the presentation contains an entry form for the data specified in node (3) and node (8), both an ISA, as these are the two ISAs whose enabling conditions evaluate to *true* at this point of time. The consultant can submit data of only one of the presented entry forms.
- When the consultant submits the data for ISA (3), the system evaluates the activation

conditions. If the system does not detect erroneous application data, a representation of DSA (5) is added to the worklist of the consultant. Otherwise, a representation of DSA (6) is added to the worklist of the consultant.

- Now the consultant sees the representation of either node (5) or node (6) in his worklist, depending on the previously taken path.[3]
- Let us assume that the consultant sees DSA (5) and activates it. The system presents to him the information from CP (2) and offers him three forms to submit to the system: A form to cancel the trip (8), a form to edit the data of the trip (3), and a form to apply for the trip (4). Now the consultant again can submit only one form, resp. one set of data, to the system. Depending on his choice the workflow continues accordingly. In case he applies for the trip (node (4)), an item (corresponding to node (7)) is created in the worklist of the reviewer.

3 Semantics of Workflow Charts using Abstract State Machines

3.1 Abstract State Machines

Some introductions to ASMs have already been written, so we only give a very brief introduction to this concept.

An ASM is a finite set of association rules of the form **if** *Condition* **then** *Updates* which transforms abstract states. The *Condition* - or *guard* - is an arbitrary predicate logic formula without free variables, which evaluates to *true* or *false*. *Updates* is a finite set of assignments of the form $f(a_1, \ldots, a_n) := v$ whose execution is interpreted as changing (or defining if undefined) in parallel the value of the occurring functions f at the indicated arguments $a_1, \ldots a_n$ to the indicated value v.

The notion of ASM *states* is the classical notion of mathematical structures where data comes as abstract objects, i.e., as elements of sets (domains, universes, one for each category of data) which are equipped with basic operations (partial functions) and predicates (attributes or relations). By default, it includes equality sign, the nullary operations *true*, *false*, and *undef* and the boolean operations.

The notion of ASM *run* is the classical notion of computation of transition systems. An ASM computation step in a given state consists in executing simultaneously all updates of all transition rules whose guard is true in the state, if these updates are consistent. For the evaluation of terms and formulae in an ASM state, the standard interpretation of function symbols by the corresponding functions in that state is used.

Functions are classified as *basic* or *derived* functions, where basic functions are part of the state, while derived function are are a kind auxiliary functions, which may vary over time but are not updatable directly by neither the ASM nor the environment.

[3] We modelled the cancel operation in a way that a consultant is presented with this option when selecting a trip for modification or for editing/correcting. We could as well model the cancel operation by making it available from the worklist.

This very short introduction is taken from the section "ASMs in a Nutshell" in [Bör04]. A full tract of ASMs can be found in [BS03]. We decided to apply ASMs because they can be seen as "a rather intuitive form of abstract pseudo-code", though based on a precise but minimal mathematical theory of algorithms, but also as "Virtual Machine programs working on abstract data" [BS03]. This obviously brings the notation very close to that of programming languages and is thus easily understandable by programmers.

3.2 Workflow Charts and Abstract State Machines

Parts of the semantics of formcharts, the predecessor of workflow charts, are specified using UML diagrams [DWL06], so it seems natural to draw from the specification of UML in ASMs, for example from Börger et al. [BCR00], [BCR04], who specify UML 1.3 state machines as ASMs, or from Sarstedt and Guttmann [SG07], who specify UML 2 activity diagrams with ASMs. Additionally, this specification is somewhat inspired by the specification of BPMN with ASMs as presented in [BT08a], [BT08b].

Since ASMs specify an operational semantics, we specify a machine which operates on a given workflow chart specification, a WFCAGENT[4]. In addition, we assume an environment which provides the events an actor can cause, namely those for selecting an element from the worklist and submitting data.

As workflow charts relay the responsibility for synchronising different execution paths of the workflow to the interplay of side effects and activation conditions [Dra10], which need to be specified by the designer of the workflow model, no general synchronisation semantics is currently given for workflow charts in [Dra10]. If one needs a specific synchronisation behaviour, the desired behaviour can be modelled as part of the guard conditions of the associations (see section 3.5 on the next page).

3.3 The Environment

The ASM we will specify later on will operate in an environment, which we will specify here. As the environment is not our main focus, we will specify elements of the environment only as detailed as necessary.

Actor. An actor is someone who activates tasks from the worklist and processes them. We will model actors as agents and define a set *Actors* which contains all defined actors.

Worklist. The worklist stores all tasks available for activation. Tasks in the worklist are associated with actors. We define the worklist as follows:

$worklist : WfCAgent \rightarrow List\ of\ DSA\_Nodes$

UI. We assume that there is a UI component available which displays information to the actor and receives input from him.

[4] This indicates that we will use ASM agents as ASM of choice. More on that later on.

3.4 Nodes

Workflow charts specify tripartite finite directed graphs whose nodes belong to the abstract set *Nodes*. The set *Nodes* is partitioned into a set for CP, ISA, and DSA nodes. The corresponding subsets of *Nodes* are *CP_Nodes*, *ISA_Nodes*, and *DSA_Nodes*.

CP_Nodes are of the form $node(type, reporting\_action)$, where the parameter *type* denotes the type of the node and *reporting_action* denotes the action used to provide the data to be reported.[5]

ISA_Nodes are of the form $node(type, action)$, where *type* denotes the type of the node and *action* denotes the function modifying the system state.

DSA_Nodes are of the form $node(action)$, where *action* has the same meaning as with ISA nodes.

We understand actions as used here as pieces of executable code which depend on the specific workflow instance. In a final system these actions thus could and would be executed in the system context while here they serve as generic place-holder which denote when the actions are to be executed.

3.5 Associations

Workflow charts have two types of associations although they are drawn identically in the diagrams: associations which are triggered by events[6], as those from DSAs to CPs or from ISAs to DSAs, and associations with a more structural aspect, as those from CPs to ISAs. Both types are modelled with one set named *Associations*. We could partition that set according to the types given above but for our purpose the unpartitioned set suffices.

Associations are of the form $assoc(source, targets, guards)$, where the parameter *source* represents the source of the association, *targets* represents a finite sequence of target nodes, and *guards* is a sequence of the same length as *targets* of Boolean expressions which guard the association from *source* to *targets*. Guards may evaluate information available in the environment as well as the available input data.

3.6 Additional Functions for Nodes an Associations

For the parameters of each type of node and association (e.g., *action* for *Nodes*, *guard* for *Associations*), we use a static function *param* which applied to the related elements yields the corresponding parameter. For example, the static function $action(node)$ yields the

[5] This action is not mandatory but it seems appropriate that there is an action preparing data to be reported. This action does not change the system state.

[6] These events are usually caused by the actor by selecting an item from the worklist.

action associated to the given node, *guard*(*association*) yields the guard of an association, etc.

In addition, we define a function *ActorOfDSA* which is defined as follows:

$ActorOfDSA : DSA\_Nodes \rightarrow Actors$

The task of the *ActorOfDSA* is to store which DSA is associated with which actor. This function could be seen as part of a rudimental authorisation system.

3.7 The WfCAgent ASM

The WFCAGENT ASM represents an actor who chooses tasks from the worklist. The WFCAGENT is implemented as an ASM agent, thus a special function *self*, denoting the currently active agent, is available. Multiple agents can process multiple items of the worklist.

The main rule of a WFCAGENT has basically two states for "selection" of a worklist item and "dataInput" for processing the actor-supplied data. Associated with these states are the following events:

$Event = \{TaskFromWorklistSelected, DataSubmitted\}$

The ASM uses an *event*-function as specified in [BG94]:

$event : Daemon \rightarrow Event$

The *Daemon* responsible for emitting the events is out of scope of this paper.

In addition, we define two *in*-functions, *SelectedNode* and *DataInput*. The *Daemon* will provide by means of the *SelectedNode* function the DSA or ISA node resp. which node corresponds to the selection from the worklist or which has been activated by data input. In case of the event *DataSubmitted*, *DataInput* will contain the data associated with the event. The function is defined as:

$DataInput : Event \rightarrow DATA*$

Finally, we define the control states of the agent:

$CtlState : WfCAgent \rightarrow \{waitingForSelection, waitingForDataInput\}$

The agent will start in the $CtlState = waitingForSelection$.

Now we can define the main rule of the WFCAGENT:

WFCAGENT =
  if $event = TaskFromWorklistSelected$
    $\wedge$ $(CtlState(self) = waitingForSelection)$ **then**
      PROCESSCLIENTPAGE(*SelectedNode*)
      $CtlState(self) := waitingForDataInput$

if *event* = *DataSubmitted*

$\wedge$ (*CtlState*(*self*) = *waitingForDataInput*) **then**

PROCESSDATAINPUT(*SelectedNode*)

CtlState(*self*) := *waitingForSelection*

To process the selected node from the worklist, we need to know which CP node is implicitly selected with the DSA node. Thus, we define a function *EnabledNodes* which yields the enabled CP as follows:

$$\begin{aligned}
&\mathit{EnabledNodes}(\mathit{source}, \mathit{sourceNodeSet}, \mathit{targetNodeSet}) = \\
&\quad \{\mathit{target} \in \mathit{targetNodeSet} \mid \\
&\qquad \mathit{source} \in \mathit{sourceNodeSet} \wedge \\
&\qquad \exists_{a \in \mathit{Associations}} \mathit{source}(a) = \mathit{source} \wedge \mathit{target}(a) = \mathit{target} \wedge \mathit{guard}(a)\}
\end{aligned}$$

To compute the CP with this function we call it in the following way:

EnabledNodes(currently active DSA, set of all DSAs, set of all CPs)

Processing of the resulting CP node is simply done by evaluating all enabling conditions for the corresponding ISA nodes and presenting a UI to the actor which contains the input fields corresponding to the types of the enabled ISA nodes. PRESENTUI is an additional ASM not specified here which is responsible for supplying the UI for the actor.

What we need to know is which ISAs are enabled by their conditions. We can use the function *EnabledNodes* for this as well by calling it with these parameters:

EnabledNodes(currently active CP, set of all CPs, set of all ISAs)

Now we can specify the macro PROCESSCLIENTPAGE:

PROCESSCLIENTPAGE(DSA) =

seq

action(*Dsa*)

cp := *EnabledNodes*(*Dsa*, *Dsa_Nodes*, *CP_Nodes*)

PRESENTUI(*cp*, *EnabledNodes*(*cp*, *CP_Nodes*, *ISA_Nodes*))

endseq

In order to wait for any action to be finished before a UI is presented, the statements of this macro are executed in sequential order.

In the second state of the WFCAGENT we need to process the data the environment resp. the actor has provided. The environment provides us with the selected node – which corresponds to an ISA node – and with the submitted data in *DataInput*.

We need to know which activation conditions are enabled, and again we can utilize the function *EnabledNodes* for this as well by calling it with these parameters:

EnabledNodes(currently active ISA, set of all ISAs, set of all DSAs)

The macro *ProcessDataInput* can then be written as follows:

PROCESSDATAINPUT(ISA) =
  seq
    action(*Isa*)
    forall *dsa* ∈ *EnabledNodes*(*Isa*, *ISA_Nodes*, *DSA_Nodes*) **do**
      add dsa to worklist(*ActorOfDSA*(*dsa*))
      where "add *x* to *y*" adds the DSA as possible task to the worklist
        of the corresponding agent
    endseq

We assume that processing of the via the function *DataInput* provided input happens in *action*(*isa*).

Note that a sequence for the action and the insertion of the next items into the worklist is specified this way. Thus, the action is completed before new items are added to the worklist. The insertion of worklist items can happen in parallel again.

4 Implementation

We implemented the specified semantics using CoreASM [SoCS11]. Due to space limitations we only give a short overview over the implementation, which is divided into two major parts: One part simulating actors and actor input – the ACTOREMULATORAGENT – and the other part being the workflow processor – the WFCAGENT. Both parts are modelled as agents, which are already supported by CoreASM via a plug-in.

The ACTOREMULATORAGENT basically waits for elements in the worklist. Then it randomly selects an element from the worklist of the active actor and submits this information to the WFCAGENT. The WFCAGENT computes the CP and all active ISAs. This information is submitted to the ACTOREMULATORAGENT to choose an ISA from. The ACTOREMULATORAGENT randomly chooses an ISA (this is the part where information would be submitted to the workflow system in a real implementation) and tells the WFCAGENT the chosen ISA. The WFCAGENT uses this choice to compute the new elements which should be added to the worklist.

5 Related Work

Formcharts as state history diagrams have been introduced in [DWL06]. State history diagrams are state transition and class diagrams at the same time. Although formcharts are the precursor to workflow charts, no precise formal semantics has been given in [Dra10]. Furthermore, formcharts show limitations regarding workflow execution as they only support single-user scenarios of two-staged human computer interaction (HCI) without concurrency [BDLW05]. Therefore, workflow charts extend formcharts with workflow semantics to develop an integrated specification for workflow definition and dialogue programming.

Another way to specify workflow charts would be Petri nets [Mur89], as they are used, for example, to specify the formal semantics of BPMN [DDO07]. There the proposed mapping from BPMN to Petri nets lacks features which coincide with the limitation of Petri nets that in turn motivated the design of YAWL [vdAtH05]. YAWL is a workflow definition language that extends Petri nets with a number of high-level features. In addition, modelling with Petri nets can become very complex very fast, as can be seen in [Tak08], where a relatively simple transaction in a travel agency's business process leads to a very complex Petri net representation. Another way to describe the semantics would have been using Communicating Sequential Processes (CSP). This has been done for BPMN in [WG08].

Finally, ASMs have been successfully used to specify the semantics of BPMN in [BT08a], [BT08b], and [BS11]. The approaches present extensive specifications trying to cover all aspects of the BPMN standard. This work on formalising workflow charts aims at specifying a plain workflow definition language, which includes exact semantics that allows generating executable code from formal descriptions of process-oriented enterprise applications.

6 Conclusion and Future Work

The semantics of workflow charts using Abstract State Machines has been introduced. The work presents a typed approach to workflow specification to improve flexibility in business process technology by proposing a concept for integrating workflow definition and dialogue programming that is also open to business process modelling. Workflow charts can be used as a technology-independent, conceptual modelling language for planning and documenting submit/response-style systems [ADG10]. Hence, workflow charts represent a platform independent model in the context of the model driven architecture community and add value by grasping the essential structure of a workflow system. By elaborating a programming language for specifying dialogue constraints, side effects and the type system, workflow charts can be exploited as a domain-specific language, i.e., a high-level programming language that overcomes the artificial separation of workflows and dialogues in business process management suites. Workflow charts extend formcharts [DW04] that represent a sufficient basis for the executable specification and the resulting description using Abstract State Machines provides a precise operational semantics for specifying human-computer interaction.

The proposed concept includes the support of actors and role models as well as the introduction of the worklist paradigm in order to present actors with their currently enabled tasks. The formalisation contributes to an exact semantics for reasoning about workflow charts and simulating models without the need to write a conventional prototype. Furthermore, having such a precise formal specification avoids common problems such as problems with OR-joins or further unclarities in the semantics of business process modelling languages. It is also possible to specify sub-workflows in workflow charts. As the semantics of sub-workflows is currently not clearly specified in the original work, a specification of the semantics regarding this aspect will be addressed later on. Additionally, the semantics of synchronisation could be specified explicitly.

Acknowledgement. The publication has been written within the project "Vertical Model Integration (VMI)" which is supported within the program "Regionale Wettbewerbsfähigkeit OÖ 2007-2013" by the European Fund for Regional Development as well as the State of Upper Austria. This work has also been supported in part by the Austrian Science Fund (FWF) under grant no. TRP 223-N23.

References

[ADG10] C. Atkinson, D. Draheim, and V. Geist. Typed Business Process Specification. In *Proc. of the 14th IEEE int. Enterprise Distributed Object Computing Conf. (EDOC 2010)*, pages 69 – 78. IEEE, 2010.

[BCR00] E. Börger, A. Cavarra, and E. Riccobene. Modeling the Dynamics of UML State Machines. In *Abstract State Machines - Theory and Applications*, volume 1912 of *Lecture Notes in Computer Science*, pages 167–186. Springer, 2000.

[BCR04] E. Börger, A. Cavarra, and E. Riccobene. On formalizing UML state machines using ASMs. *Information and Software Technology*, 46(5):287–292, April 2004.

[BDLW05] S. Balbo, D. Draheim, C. Lutteroth, and G. Weber. Appropriateness of User Interfaces to Tasks. In *Proc. of TAMODIA 2005 – 4th int. Workshop on Task Models and Diagrams for User Interface Design - For Work and Beyond*, pages 111–118. ACM Press, 2005.

[BG94] E. Börger and U. Glässer. A Formal Specification of the PVM Architecture. In *IFIP Congress (1)*, pages 402–409, 1994.

[Bör04] E. Börger. The ASM Ground Model Method as a Foundation of Requirements Engineering. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 274–275. Springer, 2004.

[BS03] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 1 edition, June 2003.

[BS11] E. Börger and O. Sörensen. BPMN Core Modeling Concepts : Inheritance-Based Execution Semantics. In D. W Embley and B. Thalheim, editors, *Handbook of Conceptual Modeling: Theory, Practice and Research Challenges*. Springer, 2011.

[BT08a] E. Börger and B. Thalheim. A Method for Verifiable and Validatable Business Process Modeling. In *Advances in Software Engineering: Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, pages 59–115. Springer, 2008.

[BT08b] E. Börger and B. Thalheim. Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach. In *Proc. of the 1st Int. Conf. on Abstract State Machines, B and Z*, pages 24–38, London, UK, 2008. Springer.

[DDO07] R. M Dijkman, M. Dumas, and C. Ouyang. Formal Semantics and Analysis of BPMN Process Models using Petri Nets. Technical Report 7115, Queensland University of Technology, Brisbane, Australia, 2007.

[DN08] D. Draheim and C. Natschläger. A context-oriented synchronization approach. In *Electronic Proc. of the 2nd int. Workshop in Personalized Access, Profile Management, and Context Awarness: Databases (PersDB 2008) in conjunction with the 34th VLDB conf.*, pages 20–27, 2008.

[Dra10] D. Draheim. *Business Process Technology: A Unified View on Business Processes, Workflows and Enterprise Applications*. Springer, 1 edition, June 2010.

[DW04] D. Draheim and G. Weber. *Form-Oriented Analysis. A New Methodology to Model Form-Based Applications*. Springer, Berlin, 1 edition, October 2004.

[DWL06] D. Draheim, G. Weber, and C. Lutteroth. Finite State History Modeling and Its Precise UML-Based Semantics. In *Advances in Conceptual Modeling - Theory and Practice*, volume 4231 of *Lecture Notes in Computer Science*, pages 43–52. Springer, 2006.

[FSKdR05] H. Fecher, J. Schönborn, M. Kyas, and W.-P. de Roever. 29 new unclarities in the semantics of UML 2.0 state machines. In *Proc. of the 7th Int. Conf. on Formal Methods and Software Engineering*, ICFEM05, pages 52–65, Berlin, Heidelberg, 2005. Springer.

[FSS+11] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger. *Subjektorientiertes Prozessmanagement: Mitarbeiter einbinden, Motivation und Prozessakzeptanz steigern*. Carl Hanser Verlag GmbH & CO. KG, July 2011.

[Gei11] V. Geist. *Integrated Executable Business Process and Dialogue Specification*. PhD thesis, Johannes Kepler University Linz, 2011.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

[Nat11] Christine Natschläger. Towards a BPMN 2.0 Ontology. In Remco Dijkman, Jörg Hofstetter, and Jana Koehler, editors, *Business Process Model and Notation*, volume 95, pages 1–15. Springer, Berlin, Heidelberg, 2011.

[OMG11] OMG. BPMN 2.0 specification, final, January 2011.

[SG07] S. Sarstedt and W. Guttmann. An ASM semantics of token flow in UML 2 activity diagrams. In *Proc. of the 6th int. Andrei Ershov memorial Conf. on Perspectives of systems informatics*, pages 349–362, Novosibirsk, Russia, 2007. Springer.

[SoCS11] Canada School of Computing Science, Simon Fraser University. The CoreASM Project, 2011. http://www.coreasm.org/.

[Tak08] T. Takemura. Formal Semantics and Verification of BPMN Transaction and Compensation. In *2008 IEEE Asia-Pacific Services Computing Conf.*, pages 284–290, Yilan, Taiwan, December 2008.

[vdAtH05] Wil M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.

[WG08] P. Y. Wong and J. Gibbons. A Process Semantics for BPMN. In *Proc. of the 10th Int. Conf. on Formal Methods and Software Engineering*, ICFEM '08, pages 355–374. Springer, 2008.