THE DESIGN AND ANALYSIS OF CONTEXT-AWARE, SECURE WORKFLOW SYSTEMS

Hind Alotaibi, Hussein Zedan

Abstract. *Workflows are set of activities that implement and realise business goals. Modern business goals add extra requirements on workflow systems and their management. Workflows may cross many organisations and utilise services on a variety of devices and/or supported by different platforms. Current workflows are therefore inherently context-aware. Each context is governed and constrained by its own policies and rules to prevent unauthorised participants from executing sensitive tasks and also to prevent tasks from accessing unauthorised services and/or data. We present a sound and multi-layered design language for the design and analysis of secure and context aware workflows systems.*

Keywords: business processes, workflows systems, workflow execution
2010 Mathematics Subject Classification: 68P20

1. Introduction

The evolution of businesses is related primarily to the evolution of their environment. The environment could be of technological and/or social nature. This continual evolution necessitates the re-engineering and optimisation of existing business processes with the objective of reducing costs, delivering timely services, and enhancing their competitive advantage in the market.

The fundamental idea of workflow technology is to separate business policies from the underlaying business applications, hence enhancing the evolution of business processes and improving the re-engineering at the organization level without delving into the application details.

Workflow systems are currently being used in numerous business application domains including office automation, finance and banking, health-care, telecommunications, manufacturing and production.

We take the view that a workflow is a set of coordinated *activities* that achieve a common business objective. Activities may be carried out by humans, application programs, or processing entities according to the organisational rules relevant to the process represented by the workflow. Activities within a workflow are usually related and dependent upon one another, which in turn are specified by a set of execution constraints which play a key role in supporting various workflow specifications such as concurrency, serialization, exclusion, alternation, compensation and so on. For example, a workflow scenario for the management of a warehouse organises and controls the movement and storage of goods within a warehouse. This is achieved through the definition and processing of complex transactions, including shipping, receiving, put away, picking and issuing of goods.

Workflow execution can involve a large number of different participants, services and devices which may cross the boundaries of various organisations. This raises important issues that are related to, for example, *context-awareness* and *security*. It is therefore important to be able to specify exact rules to prevent unauthorised participants from executing sensitive tasks and also to prevent tasks from accessing unauthorised services. For example, medical scenarios will require that only authorised doctors are permitted to perform certain tasks and that only specific machines are used in those tasks. If a workflow execution cannot guarantee these requirements, then the flow will be rejected.

Furthermore, workflows can hold and manipulate various data with different security requirements and it is important to *enforce* these requirements while the data is accessed in a workflow instance. Delegations, constraints over authorisations, audit and integrity provide additional security features. We adopt a policy-based approach in which rules are specified compositionally as policies which can be analyesed and continually verified at run-time.

Currently, there is no commonly accepted model for secure workflows or even a consensus on which features a workflow security model should support. Most policy models that are available today [3, 10] are of a static nature — it is difficult to express security requirements that are dependent on time or the occurance of events. Temporal aspects of access control are especially important in domains ranging from E-business to military domain where the value of tactical information, and therefore its protection requirements, are highly dependent on time (e.g. time to mission start) and events (e.g. adversary action, or coalition formation).

Other work [4, 5] has recognised the need for more expressive security policies to capture the temporal dimension of access control, however, these models lack compositionality. By compositionality we mean that the overall security policy can be composed out of smaller policies. This is advantageous as each policy can capture specific requirements that can be validated and verified individually and are then composed to form the overall system policy. The novelty of our model is that it does not only allow for the composition along a *structural* axis, e.g. composing policies for different departments/organisations, but also along a *temporal* dimension to account for the dynamic change of policies over time and on the occurrence of events.

This paper present a novel model for the design and analysis of secure workflow in which security requirement are expressed formally in the form of sound policies and are enforced through a dedicated agent-based sub-system. The paper is organised as follows. Section 2 gives a detailed account of our computational model together with the design language **S-Flow**. A closed-layer superstructure is given Section 3 facilitating both design and analysis of workflows. The paper ends with a short review of related work (Section 4) and some reflections and future work (Section 5).

2. Computational model

This section outlines the computational model of workflow system.

2.1. Activity. Our unit of computation in a workflow system is an *activity* which describes a piece of work that contributes toward the accomplishment of a given (functional and business) goal. An activity *starts, executes* and then *terminates*. Associated with an activity is one or more *context*. Contexts can be an organisation, a device, a service (e.g. web-based service) or a computational environment. An activity starts in one context and may terminates in a different context.

Each context is governed by a set of *security policies* which are continually changing due to either the occurrence of an event and/or the passage of time. We have chosen not to make context as a first class citizen. Our rational is that externalising policies of an activity and allowing such policies to adapt to its context, we are able to model the changes in context by adequately changing policies, and also gives us the added value that policies can still be changed within a single context.

In our model, activities may be composed concurrently to produce a new activity which terminates if and only if all of its components terminate, i.e. we adopt the *distributed termination* convention. Further, without loss of generality, we assume a single clock for an instant of a workflow.

2.2. Policies. Research on policies has mainly concentrated on the development of more expressive languages and models for the specification of security and management requirements. With the increased expressiveness tool-support has been developed to assist in the specification process and also to support the analysis w.r.t. properties such as conflicts, completeness or information-flow. However, increased expressiveness affects not only the specification of policies, but also the design and implementation of concrete enforcement mechanisms. The gap between policy specification and the implementation of enforcement mechanisms can lead to insecure systems, as the overall security of the system relies heavily on the correctness of the enforcement mechanism. Typically support for the enforcement is provided together with the policy languages. These mechanisms are however mostly developed without an underlying formal model and rely solely on the developer to have matched the semantics of the policy language. To bridge the gap between policy specification and the development of concrete enforcement mechanisms we show in this paper how enforcement code that guarantees the correct enforcement of the policy can be derived.

Further, our policies may also be annimated in the first instant to give the designer some initial confidence before the derivation of the enforcement.

Within workflow systems, *access control* policies play a fundamental role. A traditional access control policy is expressed in terms of *subjects, objects* and *actions*. Subjects represent active entities, such as users and processes, that can be authenticated within the system. The system *state* is represented by objects. Objects can only be modified by the execution of actions on request of authenticated subjects. The access control policy determines whether a subject is allowed to perform an action

on an object, or not. For example in the context of web-services a service is seen as a resource that is provided within a workflow system, to which access is controlled. A service can also request other services and is actively involved in computation. In our policy model, a web-service can therefore be seen as both object and subject. The type of request made to the web-service is modelled as an action.

The security policies are enforced by an agent. Security policies describe properties that an activity must implement. Compliance with the policy is ensured by the security mechanisms that enforce the policy on the system. Security policies themselves are an invaluable asset to any organisation as they concisely express the underlying protection requirements. Access control policies are defined in terms of rules that capture access control requirements [2]. The general form of a rule is:

$$\mathit{premise} \longrightarrow \mathit{consequence}$$

The premise of a rule determines when the rule *fires* and the consequence of the rule determines the outcome of the rule, i.e. an access control decision. We follow this approach, but allow the premise of a rule to express a behaviour rather than a predicate. The intuition is that an authorisation can be dependent on the history of execution rather than only the currently observable state. This allows the expression of history dependent authorisations such as the Chinese Wall Policy [6]. The following example shows such a rule which is expressed purely in ITL.

$$\begin{pmatrix} \Diamond do(s,o,a) \wedge clientinfo(c,o) \wedge \\ sepconcern(c,c') \wedge clientinfo(c',o') \end{pmatrix} \mapsto autho^{-}(s,o',a) \tag{1}$$

Where the predicate $do(s,o,a)$ denotes that subject s performs action a on object o. The predicate $clientinfo(c,o)$ means that object o belongs to client c and where $sepconcern(c,c')$ denotes that client c and client c' are in separation of concern. The rule given in Eq. 1 then states that when a subject has at some point in time accessed information regarding a client, the same subject cannot (negative authorisation denoted by $autho^{-}$) access information about a client that is in a separation of concern relationship. The right-hand side of a rule in the security model contains either the variable $autho$, $autho^{+}$ or $autho^{-}$. This allows to express hybrid access control policies, in which both *positive* authorisation ($autho^{+}$) and *negative* authorisation ($autho^{-}$) can be expressed. In case of conflicts, i.e. a subject has both positive and negative authorisation, a conflict resolution rule ($autho$) determines the actual access decision. Eq. 2 shows a conflict resolution rule, stating that a negative authorisation takes precedence over a positive authorisation.

$$autho^{+}(S,O,A) \wedge \neg autho^{-}(S,O,A) \mapsto autho(S,O,A) \tag{2}$$

The presented policy language is a powerful tool to express assertions on the agents' behaviours. Thanks to its compositionality it allows the policy designer to decompose otherwise complex policies into small, comprehensible components. Rules in the model are complex, due to the ability to express consequences based on past

behaviour. On the other hand this expressiveness keeps rules for complex requirements small, especially when a behavioural description of a rule's premise is natural.

A simple policy is a set of *rules* that describe a condition under which a specific *action* may be taken. This condition can be a *temporal formula* describing the past behaviour of the agent.

2.3. The design language S-Flow. Our computational model is supported by a design language, known as Secure-Flow (**S-Flow**), with which we are able to design and analyse workflows.

- **Activities**
  $A ::= \ < P >: \ (PA \ ^\frown CA)$
- **Policies**
  $P ::= \ p \ ^\frown P ; Q \ ^\frown \langle w \rangle P \ ^\frown \ [w] P \ ^\frown t : P \ ^\frown P^* \ ^\frown P \rhd_w^t Q : R \ ^\frown P \| Q$
- **Primitive Activities**
  $PA ::= \ x \ := \ v \ ^\frown \mathit{skip} \ ^\frown \Delta(t) \ ^\frown \ [t_1 \ldots t_n] A \ ^\frown C \ ! \ v \ ^\frown C \ ? \ x$
- **Compound Activities**
  $CA ::= \ A_1 \ ; A_2 \ ^\frown A_1 \ \| \ A_2 \ ^\frown A_1 \rhd_t^e A_2 \ ^\frown A^* \ ^\frown g_1 \ \rightarrow A_1 \ \Box \ g_2 \ \rightarrow A_2$

The informal semantics of these constructs are as follows.

- $< P >$: A is an activity A which is governed by a set of policies p. Here we have not made the context, within which the activity is to be executed, a first class citizen. Instead we have assumed that any context information will be encoded within the policies. There is a number of rich operators that can be used to compose policies. Let p denotes a simple policy, w denotes a state formula, t a natural number, P, Q and R range over policies. The informal description of the operators follows. Policy composition can be used for the incremental development of security policies. The advantage of this approach is that small policies are easier to comprehend and verify. The compositional operators can then be used for the integration of the overall system policy.
  - $P ; Q$: Sequential composition of two policies. The system is first governed by policy P and then by policy Q.
  - $\langle w \rangle P$: The system is governed by policy P unless w holds. The state formula w can here indicate the happening of an event.
  - $[w]P$: The system is governed by policy P as long as w holds.
  - $t : P$; defines that policy P is enforced for t time-units.
  - P^* : defines an iteration of policy P.
  - $P \rhd_w^t Q : R$: Behaves like policy P until a condition w becomes true or t time-units elapse. It then changes to behave like R if w is true or like Q otherwise.
  - $P \| Q$: Both, policy P and policy Q apply at the same time.
- Activities vary from primitives to compound:

- $x := v$ describes an activity that assigns the value v to a variable x.
- $\Delta(t)$ is an activity that delays a workflow by a duration of t time.
- *skip* is a null activity.
- $[t_1 \ldots t_n]A$ describes an activity A that will take either t_1, or t_2, or t_n time units.If $n = 1$, the activity will take exactly a t time. This is a powerful construct as it provides a greater flexibility at execution time of a workflow. At run time, the scheduler will have the choice between the various deadlines which will depend on the available resources.
- $C\,!\,v$ and $C\,?\,x$ describes how activities are communicating with each others: $C\,!v$ is an activity that send the value v over the channel C; whilst $C\,?\,x$ describes an activity that receives a value over a channel C and stores it in x.
- $A_1\ ;\ A_2$. It is an activity which starts with a sub-activity A_1 and upon its proper termination, the sub-activity A_2 begins. Here we assume that the sequential composition is instantaneous. However, to model time consumption, we may utilise the delay activity $\Delta(t)$.
- $A_1 \parallel A_2$. This the parallel composition activity. The activities A_1 and A_2 execute concurrently and the whole activity terminates if and only if both sub-activities terminate.
- A^*. This describe an activity in which A repeats itself.
- $A_1 \rhd_t^e A_2$. This is an activity that starts with the activity A_1. Then A_2 starts if either a t time units has elapsed or an event e has occurred whichever happens first. The t and e are optional :
  - $A_1 \rhd A_2$. This is an activity which is equivalent to A_1.
  - $A_1 \rhd^e A_2$. This is an activity which starts with A_1 and only if an event e occurs then A_2 takes over, i.e it acts as an interrupt.
  - $A_1 \rhd_t A_2$. Here the activity starts with A_1 and after a t time units elapsed activity A_2 starts. Notice here that t is not related to the execution time of A_1. For example, if $t = 0$ then $\rhd^t A_2$ is equivalent to A_2. However, if t is greater than the execution time of A_1, then its termination has to be delayed until t elapsed, then A_2 starts.
- $g_1 \rightarrow A_1 \,\Box\, g_2 \rightarrow A_2$. This activity is to model non-deterministic choice. The guards g_1 and g_2 are boolean expression which are evaluated at the start of the activity. If both guards are false, the activity is failed.If both are evaluated to true, then one of the associated subactivities is chosen at random and executed.

Let us consider the following simple but illustrative example.

Example 1. A travel reimbursement processing workflow (*Reimb*) can typically has four activities: A_1 - Preparation of the claim; A_2 - Approving claim; A_3 - Issuing a check; and A_4 - Notification of declination.

There is obvious dependencies between these activities. For example A_2 can not start before A_1 properly terminates and that either of A_3 or A_4 can only starts after

the proper termination of A_2. There are also security policies governing each of these activities. *Only a customer can complete the claim*, *A supervisor is the only person who can approve the claim* and *A clerk can notify customer and/or issuing check* are just few security requirements constraining the workflow.. Using **S-Flow**, we can design *Reimb* as follows.

Let *Status* be a state variable denoting the status of the claim and let C_1, C_2 and C_3 be communication channels between the activities. The whole workflow system can be expressed as:

$$Reimb \;\widehat{=}\; A_1 \,;\, [20]A_2 \;;\; (\, Status \;\rightarrow\; A_3 \,\Box\, \neg Status \;\rightarrow\; [5]A_4)$$

Note here that the Supervisor in A_2 must make a decision within 20 time units and that the Clerk should notify the customer by the decision within 5 time units. Here we only give a full specification of A_1:

$$A_1 \;\widehat{=}\; <(S \;=\; Customer \;\wedge\; O \;=\; ClaimForm \;\wedge\; a \;=\; fill)$$
$$\mapsto\; autho(S,O,a) >:\; Complete \;;\; C_1!C_form$$

I.e. only a customer can fill the claim and, once completed, it is sent via channel C_1.

2.4. Formal Semantics. In security critical workflows it is paramount that the language to express design and policies has a sound and formal basis that can be used for the analysis and proof of properties such as conflicts, completeness or information-flow. The formal semantics of **S-Flow** is given in a specification-oriented style expressed in ITL. An important reason of choosing ITL is the availability of an executable subset of the logic, known as Tempura [7]. The Tempura interpreter takes a Tempura formula and constructs the corresponding sequence of states. The use of ITL, together with its subset of Tempura, offers the benefits of traditional proof methods balanced with the speed and convenience of computer-based testing through execution and simulation. The entire process can remain in one powerful logical and compositional framework. We recall that our semantics domain is interval-based. An interval is defined to be a (in)finite nonempty sequence of states $\sigma \;\widehat{=}\; \sigma_0\sigma_1\ldots$, where each state σ_i is a *value assignment* which associates an integer number with each variable:

$$\sigma_i \in \Sigma \;\widehat{=}\; Var \rightarrow \mathbb{N}.$$

We also denote by Σ^+ and Σ^ω the sets of finite intervals and the set of infinite intervals respectively. The *semantics of an activity A* is a function

$$\mathcal{M}[\![A]\!] \in (\Sigma^+ \cup \Sigma^\omega) \rightarrow \{\mathrm{tt}, \mathrm{ff}\},$$

defined inductively on the structure of activities as follows.

$$\mathcal{M}[\![< P >:\; (PA\,\bar{}\,CA)]\!](\sigma) \;\widehat{=}\; P \wedge\; (\mathcal{M}[\![PA]\!](\sigma) \;\vee\; \mathcal{M}[\![CA]\!](\sigma))$$

$$\mathcal{M}[\![x \;:=\; v]\!](\sigma \;\widehat{=}\; \bigcirc(x) \;=\; v$$
$$\mathcal{M}[\![\mathsf{skip}]\!](\sigma) \;\widehat{=}\; \bar{\sigma}{=}1$$

$$\mathcal{M}[\![\Delta(t)]\!](\sigma) \;\widehat{=}\; \bar{\sigma}{=}t$$

$\mathcal{M}[\![[t_1 \ldots t_n]A]\!](\sigma) \;\widehat{=}\; \exists\, i \in \{1 \ldots n\} \cdot \overline{\sigma} = t_i \wedge \mathcal{M}[\![A]\!](\sigma)$

$\mathcal{M}[\![C\,!\,v]\!](\sigma) \;\widehat{=}\; \mathit{fin}\ c = v$
$\mathcal{M}[\![C\,?\,x]\!](\sigma) \;\widehat{=}\; \mathit{fin}\ x = c$

$\mathcal{M}[\![A_1\ ;\ A_2]\!](\sigma) \;\widehat{=}\; \mathcal{M}[\![A_1]\!](\sigma)\ ;\ \mathcal{M}[\![A_2]\!](\sigma)$
$\mathcal{M}[\![A_1 \parallel A_2]\!](\sigma) \;\widehat{=}\; \mathcal{M}[\![A_1]\!](\sigma) \wedge \mathcal{M}[\![A_2]\!](\sigma)$

$\mathcal{M}[\![A_1 \rhd_t^e A_2]\!](\sigma) \;\widehat{=}\; (e \wedge \mathcal{M}[\![A_2]\!] \vee (\mathcal{M}[\![A_1]\!](\sigma) \wedge (\Box\neg e) \wedge \mathit{len} < t) \vee ((\mathcal{M}[\![A_1]\!](\sigma) \wedge (\Box\neg e) \wedge \mathit{len} < t); \mathit{skip}; (\mathcal{M}[\![A_2]\!](\sigma) \wedge e)) \vee ((\mathcal{M}[\![A_1]\!](\sigma) \wedge (\Box\neg e)$,provided $0 < t < \infty$.

$\mathcal{M}[\![(A^*)]\!](\sigma) \;\widehat{=}\; (\mathcal{M}[\![(A)]\!])(\sigma)^*$
$\mathcal{M}[\![g_1 \rightarrow A_1 \,\Box\, g_2 \rightarrow A_2]\!](\sigma) \;\widehat{=}\; (g_1 \wedge \mathcal{M}[\![A_1]\!](\sigma)\,) \vee (g_2 \wedge \mathcal{M}[\![A_2]\!](\sigma)) \vee \mathit{false}$

3. Layers: a design principle

Two classes of properties are of particular interest when we consider any computer systems. These are *safety* and *liveness* properties. For large concurrent systems (such as workflows), current techniques are complex when applied to highly concurrent systems where context can also vary during its execution.

We present the notion of **Information-closed** layer as (a) **design principle** of secure workflow applications. (b) a **programming construct** which can offer a linguistic support for information flow security and (c) an efficient approach for both **static and dynamic analyses** for information security with an application such as that found in workflows. The concept is sound in the sense that it has a solid foundation in mathematics and verification.

3.1. Layer Construction. Without lose of generality, a worflow system, *Sys*, with n sequential activities, takes the general form

$$Sys \,\widehat{=}\, A_1 \parallel A_2 \parallel \ldots \parallel A_n \quad 2\text{cm (1)}$$

where

$$\forall i, \exists j \,.\, A_i \widehat{=} {S_1}^i\ ;\ {S_2}^i\ ;\ \ldots ;\ {S_j}^i$$

and that each ${S_k}^i$ is a primitive activity.

3.2. Communication Layers. Analysing a concurrent workflow is hard. To facilitate this process we present a transformation theory that transform a given concurrent workflow into what we call **quasi-parallel** workflow. This provides a *super* structure over the usual activity-based structure.

let us consider the system given in (1) and let each A_i be

$$A_i \,\widehat{=}\, {S_1}^i\ ;\ {S_2}^i\ ;\ \ldots\ ;\ {S_j}^i\ .$$

Let us also assume without lose of generality that all processes have the same number of primitive processes. The rationale is that the shorter process can be padded

by as many as required with *Skip* activity without altering its original semantics. For example, the activities A_1, A_2 and A_3 are all semantically equivalent:

$A_1 \widehat{=} \quad {S_1}^1$
$A_2 \widehat{=} \quad Skip\ ;\ {S_1}^1$
$A_3 \widehat{=} \quad {S_1}^1\ ;\ Skip$

Let us consider a simple concurrent system, S_1:

$S_1 \widehat{=} \quad A_1 \parallel A_2$
where

$A_1 \widehat{=} \quad x := a\ ;\ C?y\ ;\ z := x + y$
$A_2 \widehat{=} \quad C!w$

We can have a number of superstructures generated from S_1. For example, here are just two of such superstructures.

$S_2 \widehat{=} \quad L_1\ ;\ L_2\ ;\ L_3$
where

$L_1 \widehat{=} \quad x := a \parallel Skip$
$L_2 \widehat{=} \quad C?y \parallel C!w$
$L_3 \widehat{=} \quad z := x + y \parallel Skip$

The other is

$S_3 \widehat{=} \quad L_1\ ;\ L_2\ ;\ L_3$
where

$L_1 \widehat{=} \quad x := a\ ;\ C!w$
$L_2 \widehat{=} \quad C?y \parallel Skip$
$L_3 \widehat{=} \quad z := x + y \parallel Skip$

We may observe here that in $L2$ of S_2, the communication over channel C starts (via $C!w$) and completed (via $C?y$) within the same layer. In this case we call the layer **communication-closed**. This case was not the same in L_2 of S_3, in which the communication started in L_2 and was completed in L_3.

Definition 3.1. A layer is called communication-closed if a communication is initiated and resolved in the same layer.

Communication-closed layers can be seen as atomic transaction providing a natural mechanism for designing fault-tolerant workflow system.

An important observation of note is that which of these superstructures (which we will call **layer** structure) is semantically equivalent to the original process structure?

Theorem 3.2. A distributed workflow system $Sys \widehat{=} A_1 \parallel A_2 \parallel \ldots \parallel A_n$, such as $A_i \widehat{=} {S_1}^i\ ;\ {S_2}^i\ ;\ \ldots ;\ {S_j}^i$ can be decomposed to a semantically-equivalent layer superstructure, $Sys_1 \widehat{=} L_1\ ;\ L_2\ ;\ \ldots L_n$, where $L_i \widehat{=} {S_1}^i \parallel {S_2}^i \parallel \ldots \parallel {S_j}^i$, if and only if, each L_i is communication-closed.

This result lead us to make the following observations:

- Layer structure will facilitate and simplify the process of verification of safety and liveness properties. This is because the new structure transforms a highly concurrent workflow into a set of quasi-sequential sub-workflows which each can be analysed using currently available techniques used in sequential systems.
- At a design stage, the superstructure allows us to decompose security requirements across the activities. This ensures *information flows* securely and that the boundary of the layer acts as a point in which information does leak from a layer with security level to one with a lower level.

4. Related work

There are various approaches to deal with adaptability of workflows which can be classified as run-time, automated and instance-based. However, none of them deal with adaptation at a secure context level.

In [11], BPEL workflows are extended to support policies to look up, select and bind partner services compliant with a port type defined within an activity in a workflow definition. Such a *find-and-bind* approach is not, unlike ours, context-based neither compositional. In [8], a specific extension of the BPEL to allow adaptation of interfaces to human operators. The approach and its accompanied architecture (known as PerCollab) is still design-time, manual and evolutionary; it differs from approaches such as [1, 9] due to its purely vertical, refinement-based nature. Similarly, the work in [16] supports adaptation in declarative terms, focusing on the notion of inheritance In [13], a minimal set of rules is used in order to enforce changes in workflow instances. Such rules are essentially (though not exclusively) based on the idea of refining the running code with specific instances. Such rules are designed in a way that it is possible to formally guarantee that the adapted workflows are persistently consistent and correct. The approach is not tailored to a specific work-flow language, but it is defined on a generic formal model of workflow. In [14], the focus is shifted to a suitable subset of the BPEL language. In both cases, the approach is run-time and instance-based. However, the approach lacks the ability to change policies according to change in contexts. Furthermore, a methodology to define distributed self-healing applications was devised. In such a setting, a first question they address is that of diagnosability and reparability testing of workflows. A second one, more relevant in our context, regards the enactment of self-healing mechanisms. In this sense, an architecture is proposed that enriches the run-time support for the sake of diagnosing and fault repairing, in order to enact self-healing of process instances, and also to guarantee that certain QoS criteria are obeyed. The work is rather at an infrastructural level, and concerns adapting single run-time workflow instances. There are approaches that, while still related to (built-in) adaptation, take different views and focus on different, and sometimes specific, issues. In the [12] approach, the focus is rather on guaranteeing the soundness, based on well-founded semantics, of adaptive workflows inside some given workflow management system. To achieve

this, the authors inject workflows with semantic constraints of specific forms, paving the way to their formal verification, both considering run-time instances and adapted schemata of workflows. In [15], the focus is to guarantee the alignment between adaptable process schemata and their run-time instances. In particular, changes to a schema must reflect to the process instances, and because of misalignments, conflicts of various kinds may arise. The work identifies methods to identify such conflicts, based on execution equivalence and structural comparison.

5. Conclusion

Current workflows cross boundaries of organisations and may require services which reside on different devices and/or operating on variety of platforms, security issues have become crucial in their design and analysis. This is in addition to the fact that workflows have become *context-aware*. Workflow systems must therefore enforce the variety of security requirements as described by the current context in which the workflow is executed. This paper described an approach for designing, analysing workflows particular attention to security requirement and their distributed enforcement.

There are many outstanding issues that need to be considered. Fundamental is the management of security polices in the presence of context changes. this may require appropriate changes in the computational model to cater for mobility and dynamic change management. we shall consider this in another forthcoming paper.

6. Acknowledgement

The authors would like to thank colleagues in the Software Technology Research Laboratory (STRL) for all the fruitful discussion and constructive comments, specially, Drs. A. Cau, F. Siewe and B. Moskowzki.

H. Alotaibi particularly would like to acknowledge the financial support from the Government of the Kingdom of Saudi Arabia.

The authors wish to acknowledge the support of the National Science Fund (Research Project Ref. No. DO02-149/2008) as well.

References

[1] Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Conferences*, volume 1, pages 291–308, 2006.

[2] Martín Abadi. Logic in Access Control. In *Proceedings of the 18th Annual Symposium on Logic in Computer Science (LICS'03)*, volume 15, pages 228–233, Ottawa, Canada, June 2003. IEEE Computer Society Press.

[3] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A Calculus for Access Control in Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[4] Steve Barker and Peter J. Stuckey. Flexible Access Control Specification with Constraint Logic Programming. *ACM Transactions on Information and System Security*, 6(4):501–546, November 2003.

[5] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3):191–233, 2001.

[6] David Brewer and Michael Nash. The Chinese Wall Policy. In *IEEE Symposium on Research in Security and Privacy*, pages 206–214, Oakland, California, USA, May 1989. IEEE.

[7] Antonio Cau. A not so short introduction to ITL, September 2004.

[8] Dipanjan Chakraborty and Hui Lei. Pervasive enablement of business processes. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*.

[9] Michael George and Jon Pyke. Dynamic process orchestration. In *White paper, Stallware PLC*, 2003.

[10] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[11] Buchmann A. Karastoyanova, D. Refflow: A model and generic approach to flexibility of web service compositions. In *Proceedings of International Conference on Information Integration and Web-based Applications and Service (iiWAS 2004)*, pages 27–29, 2004.

[12] S. Rinderle L. T. Ly and P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. volume 64, pages 3–23, 2008.

[13] Peter Dadam Manfred Reichert. Adeptflex - supporting dynamic changes of workflows without loosing control. 10(2):93–129, 1998.

[14] Manfred Reichert and Stefanie Rinderle. On design principles for realizing adaptive service flows with bpel. In *Proc. EMISA 2006*, pages 133–146, 2006.

[15] Manfred Reichert Stefanie Rinderle and Peter Dadam. On dealing with structural conflicts between process type and instance changes. In *Proc. 2nd. Int'l Conf. Business Process Management (BPM'04)*, volume 3080 of LNCS, pages 274–289, 2004.

[16] H.M.W. Verbeek P.A.C. Verkoulen W.M.P. van der Aalst, T. Basten and M. Voorhoeve. Adaptive workflow: An approach based on inheritance. In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management*, pages 36–45, 1999

Hind Alotaibi, Hussein Zedan
Software Technology Research Laboratory
De Montfort University, Leicester, U.K.
e-mail: hind.alotaibi2@myemail.dmu.ac.uk, hzedan@dmu.ac.uk