# Towards Feature Interactions in Business Processes

Stephen Gorton and Stephan Reiff-Marganiec

Department of Computer Science
University of Leicester
University Road
Leicester LE1 7RH
United Kingdom
*email: {smg24,srm13}@le.ac.uk*

**Abstract.** The feature interaction problem is generally associated with conflicting features causing undesirable effects. However, in this paper we report on a situation where the combination of features (as policies) and service-targeted business processes yields non-negative effects. We consider business processes as base systems and policies as a feature mechanism for defining user-centric requirements and system variability. The combination of business processes and a diverse range of policies leads to refinement of activities and possible reconfiguration of processes. We discuss the ways in which policies can interact with a business process and how these interactions are different from other approaches such as the classical view of POTS or telecommunications features. We also discuss the conflicts that can arise and potential resolutions.

## 1 Introduction

Feature Interaction [5] was first identified as a problem in telecommunications, where additional units of functionality (features) would interfere with each other and cause unpredictable behaviour. Telecommunications have become increasingly complex, including the birth of Internet Telephony services. Removing the "Telephony" part, feature interaction has also been identified in Web Services [38–41].

Web Services [1] are an implementation of Service Oriented Architecture (SOA), where the design of systems shifts from overall development to the orchestration of services. At a more abstract level, workflows are used to specify business processes. Each task in a workflow represents a unit of activity and can be completed by using a service. Business Process Management (BPM) research has often reported that it is paired well with SOA to produce flexible business software solutions (e.g. [26]).

Policies are generally agreed as information that can modify a systems behaviour at runtime, without the need for recompilation or redeployment [19]. In our work, we use policies to express essential requirements and system variability

by combining them with workflows [12] – the latter essentially means treating policies as features.

The combination of policies and workflows in the context of Service Oriented Computing (SOC) can lead to feature interactions. What has not been studied is the nature of these interactions. This paper discusses and classifies the types of interactions that can occur, while also showing how the interactions occurring here are different from those in the more traditional domain of telecommunications, and even the more recent domain of policies.

Throughout the paper the terms *conflict* and *interaction* are used with specific meaning. Interaction will be used to describe points of contact between workflows and policies; interactions are by and large desirable and hence a positive thing. Conflict will be seen as issues that arise between policies or when selecting services and are usually undesirable. The only exceptions to this are when we consider the traditional area of policy conflict, we will use the term *policy conflict*; when considering traditional *feature interaction* we will use that term.

*Overview.* The remainder of this paper is structured as follows: Section 2 presents some background material on workflows, Service oriented Computing and policies. Sections 3 and 4 contains the two major contributions. First we present an analysis of the differences and similarities of feature interaction in its more traditional contexts and in the context of workflows, and then consider the types of interaction that can occur in the new setting with suggestions towards solutions. The paper is rounded off with related work in section 5 and a discussion in section 6.

## 2 Background

There are three main ingredients to our work: workflows, Service Oriented Computing (SOC) and policies. Each serve a distinct purpose. Workflows describe the basic process model defining the main functionality. SOC is the foundation of the implementation. Policies augment the workflow to customise it to a particular user's preference. The combination of the three provides us with the Service-Targeted, Policy-Driven Workflow approach that we call StPowla. An overview showing the relations between the StPowla elements is shown in Fig. 1.

Policies are considered an overlay mechanism (including monitoring and enforcement) for business processes. Business processes are the workflow models viewed from the business or application domain. Often, the authors of such workflows are the end users, i.e. business analysts rather than software engineers. Business process models may then be mapped to more precise service models (e.g. SRML [8]) and then to concrete orchestration models. These are then mapped to services via platform-independent middleware.

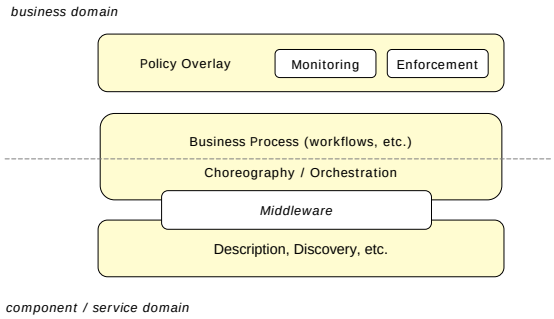


**Fig. 1.** StPowla overall architecture.

### 2.1 Workflows

A workflow is a connected graph of activities, or *tasks*. Each task represents a unit of work that contributes to a wider goal. Workflows are the accepted mechanism for describing business processes, where a defined sequence of tasks contribute to the satisfaction of a business objective.

Workflow description languages exist to describe processes in either code-based or graphic-based notations. Examples of the former include YAWL [33] and ebXML [32], while examples of the latter include BPMN [13] and UML Activity Diagrams [7]. In terms of SOC, BPEL [24] is the most widely accepted business process language, in that it can describe the process and orchestrate a number of services into the process.

In our approach, a workflow is a core business model containing enough functional requirements to map each task to a service (Fig. 2). At runtime, a task is automatically assigned a service. This assignment includes the discovery of the service, binding to it and invoking it (thus each task is a distinct computation from all other tasks). A workflow should have enough information attached to it to run successfully on its own.

Policies are used to alter this process through either refinement or reconfiguration of the workflow. This kind of intervention is required to either maintain a current state (e.g. keep costs to a specified level) or to execute a different path of processing.

The key difference between a workflow for a business process and a workflow for another purpose (e.g. telecoms or home networks) is time. Business processes execute over a long period of time (perhaps hours or days). They include error handling or *compensation* actions for the recovery of the workflow in the case of failures. Also, policies may not have an effect immediately. Once triggered, they may take a period of time before the effects become evident.

### 2.2 Service Oriented Computing

In SOC, software exists as separate entities, developed in isolation as services that are loosely coupled, platform independent, composable and based on open

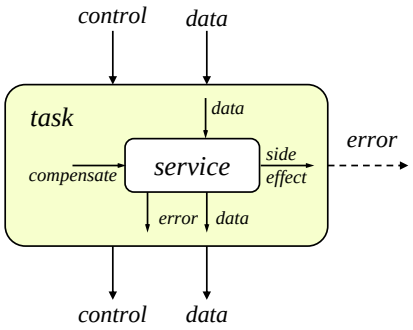


**Fig. 2.** Tasks and their relation to services.

standards. In addition, they may be discoverable and self-describing. A Web Service is discoverable through directory services such as UDDI [23], is self describing through WSDL [36], is composable through a variety of mechanisms (the de facto standard is WS-BPEL [24]) and is based on XML as the open standard (e.g. messaging is often done through SOAP [35]).

Services are key to this work. They are reusable software components that take part in a wider process. Our aim is to develop a policy-driven process model that is satisfied by services. Thus, the author of a process is not expected to write functional code, but rather specify enough requirements such that they can be mapped to an existing service (we note that there is the need for syntactic match-making between the process author and services).

Services provide agility to processes in that a system is no longer confined to one individual implementing component. One service can be substituted for another, provided it takes the same inputs and returns the same type of outputs. IBM's business model is based on the Service Component Architecture (SCA) [16], which is based on SOA. A client's requirements are satisfied by a composition of IBM's services (if a service doesn't exist, they create it), thus the product supplied to a client is actually a composition[1].

### 2.3 Policies

Policies are end-user defined rules for the management of a system. Our policies are either Event-Condition-Action rules (ECAs), or goals (e.g. constraint rules).

Policies are a proven integrated software management technique. They force a system into dynamic behaviour as the system must react to given rules at runtime. Policies can be added incrementally, with (theoretically) no limit on the number that can be applied at once. However we do note that the probability of policy conflict grows as the number of policies increases.

A policy conflict occurs when two or more policies contradict each other in terms of what the system is instructed to do or what state it should maintain. There are broadly three categories of conflict: goal conflicts, function conflicts

[1] Keith Goodman's (IBM) recent keynote at IM2007

and combined goal/function conflicts. A goal conflict is when two goals are in contradiction of each other. A functional conflict is when two policies state two different (non-compatible) paths of system execution. A combined conflict occurs when a functional policy chooses a system execution path that would violate a goal.

We use the APPEL policy description language [28] to define our policies. APPEL is an XML-based language, which has recently gained formal semantics via a mapping to $\Delta DSTL$ [20] (formerly APPEL only had a natural language semantics). It was developed initially as a call control language for the Internet Telephony domain, but is based on a core language with domain specialisations. In our research, we are working towards a customisation of APPEL as a policy-mechanism for service-oriented business modelling. This requires some knowledge of the target domain through ontologies.

## 3 Feature interactions in SoC workflows

Feature Interactions in the context of policies applied to workflows shows all the characteristics of traditional feature interaction, especially that they may hinder advancement of the system at runtime or at least violate user expectations. However, they differ in two significant aspects: one being an assumption about the knowledge of the main system available to policy designers and the other an assumption about the lifespan of the effect of an action. This section discusses both in more detail.

### 3.1 Details of Base System

Considering traditional feature interaction, e.g. in the domain of telecommunications, we notice that there are two fundamental components: a base system and the features. In this domain features have been written by programmers with a sound knowledge of the base system and in general one would always expect a feature deployed on a base system to work correctly in the absence of other features. This notion is fundamental in the definition of feature interaction: if a feature $f_1$ satisfies a property $\phi_1$ (written $f_1 \models \phi_1$), and $f_2 \models \phi_2$; a feature interaction is said to occur if, when the features are composed (denoted $f_1 \oplus f_2$) we do not have $f_1 \oplus f_2 \models \phi_1 \wedge \phi_2$.

We have argued [27] that in the context of policy conflict there is no explicit base system and that conflict emerges between a number of policies. This proved fruitful for addressing policy conflict in a structured way [20].

Considering workflows we are in a setting that differs from both of the above: the workflow presents a base system onto which policies are applied – however, the authors of policies do not need to be aware of the workflow (it would help if they were), as for example a business might change its overarching business policy regarding communication by email for security purposes, not realising that several of the workflows that are conducted within the business rely to some extend on email communication. It is clear that we have a number of

stakeholders in this setting, some that are involved in formulating the business process and some that are involved in formulating policies applicable to the same.

Clearly this breaks the fundamental assumption in feature interaction that a feature will operate as expected if it is put together with the base system.

### 3.2 Future Effects

In traditional telecommunications systems the effect of a feature is relatively immediate: that is when a feature gets invoked it will perform some action. This has been used extensively in the approach by Calder et al. [4] where a feature manager was exploring the next responses and would use a commit and rollback mechanism to select a solution. In particular reorderings of features were explored in their work to allow for the fact that executing A followed by B might be acceptable whereas B followed by A leads to a conflict.

Business processes differ fundamentally in this aspect in that the execution of a service might be performed over long periods of time. Using compensation actions (as described using the Sagas calculus in [3]) for workflow recovery, these business processes are regarded as long running transactions (LRTs). The effect of an LRT is that a feature that has no initial effect may have an effect sometime in the future.

Let us consider a simple market trader. Their core business process involves selling products to buyers, reflected in a workflow details of which we can omit here. The trader adds two policies to their business model: the first specifies that new stock from a supplier is ordered once a trade has occurred and the second negotiates the price with the supplier of the product that was just sold. By adding these policies to the workflow the business process can be streamlined.

An analysis of the example shows that if the reordering feature executes first, then the trader reorders supply at a previously agreed price. The second feature is then activated and a new price is negotiated. While the new price will not have any effect on this transaction (since the purchase has already been made), it will have an effect on future transactions. If the price negotiation feature executes first, then the price is renegotiated and then the reordering occurs with the new price.

This example highlights very clearly that the order of execution of the policies matters – something that was to be expected and is very much in line with the observation from [4]. However, what is novel is the fact that the result of the execution of the policies, in either order, has a lasting effect on a future transaction.

## 4 Types of Interaction

When considering workflows that are enhanced with policies in the context of service oriented computing, we can distinguish three broad types of interactions: conflicts between a number of applicable policies, conflicts between policies and

workflows as well as conflicts in the service level agreements. In this section we consider all three types and will show that they are, while of course all being interactions, different in the way they emerge and how they need to be dealt with.

### 4.1 Policy Conflict

Policy conflict occurs when two or more policies can be active at the same time and lead to conflicting actions being requested. Policy conflict has been defined in [27], where it was also pointed out that this definition is based on a specific application domain: only by considering a domain can a clear statement be made about which actions conflict. However, in addition there might be types of conflict that exist within the policies, independent of the application domain. These have been discussed in detail in [27] and we have mentioned these in the background section.

Let us consider an example: In a bank loan approval process the workflow has a task of making an offer followed by a task to vet the offer. Typical policies attached to this might be *"the vetter must be different from the offer maker"* and *"managers might make offers and vet these"*. Clearly the execution of the second task (the vetting) might be allowed or blocked depending on how the two policies are interpreted. Cases like this have been discussed in the taxonomy in [27], and we can identify elements of Roles, Domain Entities and Policy Relation here: the employees, including the manager have places in the domain hierarchy and also play specific roles (vetter, offer maker). Furthermore one can argue that the policies allowing the manager to make offers and vet them is a refinement of the more general rule of having two people involved in the process.

As the previous example shows, the conflicts between policies do not show any new characteristics, they do however continue to exist in this new domain. Detection and resolution methods fall into the categories discussed in [27], with a desire to detect and resolve as many issues at design-time but keeping in mind that this is not always possible and hence that decisions will need to be taken at runtime. Generally design-time methods will apply when the policies are under control of the same person or details of the overarching policies are known to the policy author (that is e.g. within a group or enterprise) detection involves some reasoning on a logical level (as e.g. in [20]) and resolution would involve policy redesign. However, if the workflow spans a range of businesses or the services are outsourced then detection of conflicts will only be possible by runtime methods and resolution will usually involve negotiation or some other dynamic means.

What is however interesting to note for the purpose of this paper is the aspect about domains that was not considered in the taxonomy. When considering the policies in relation to workflows, which themselves are implemented by services we obtain several levels of 'application domain': on one hand we can consider the workflow to be the application domain, on the other hand the services can be seen as the application domain (and then one could further investigate which domain the services belong to). The next two subsections address these issues respectively.

### 4.2 Policies on Workflows

A policy has the ability to manipulate a workflow in two ways. Firstly, it can *refine* the workflow by expressing further requirements for each task. An implementing service is restricted by all requirements in the task, thus the more requirements stated means that the service selection process becomes more precise and closer to the user's needs. Secondly, a policy can *reconfigure* a workflow. This involves stating rules for the insertion or deletion of process components. This second concept is explained in the following example:

*Example 1.* Consider a simple purchase process, where you request quotes from 3 suppliers and then you purchase from the cheapest. Suppose we add a policy that states "*If the quote from A is below £100, cancel the other quotes and purchase directly from A*". If the price from A comes in below the given amount, then the workflow changes (Fig. 3).

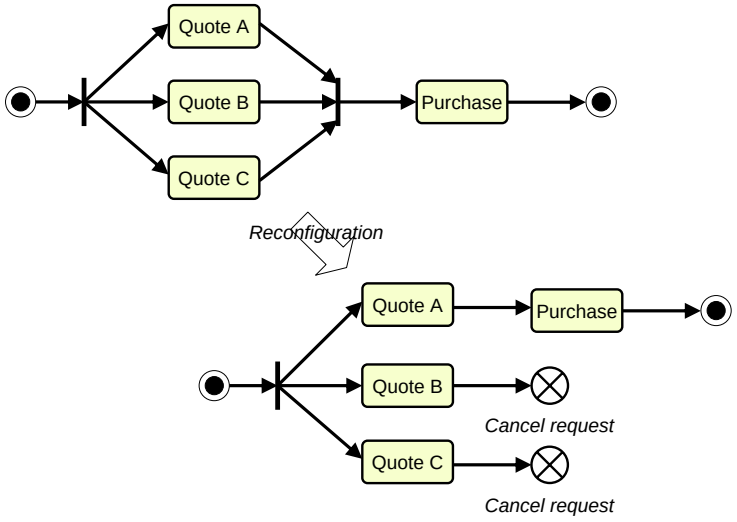


**Fig. 3.** Example 1 basic workflow

**Refinement Policies** A refinement policy can be created by multiple stakeholders. This implies that a policy can be directed at different levels of process complexity. For example, an IT director may write a policy that overarches a set of processes whereas a project team member may write a policy solely for a single task of a process. Refinement is done through policies specifying constraints over tasks through SLA dimensions[2].

[2] more information in section 4.3

This effectively enables stakeholder (or goal-based) conflicts, where different levels of stakeholders can add their own policies, without realising they conflict with others. Furthermore, there is a need to specify priorities over policies.

*Refinement Conflicts* Policy authors are already able to specify modalities (must, should, prefer and their negations), but in the case of two conflicting policies that both have the same modality (*must* in the worst case), then a resolution is required.

Possible solutions include the prioritisation of stakeholders: higher stakeholders have priority over lower stakeholders (e.g. directors over project team members). This method requires robust selection that will ensure that only specific stakeholders are allowed to create policies. Even then, policies should be agreed in advance and published to other stakeholders. In this situation, only generic policies (i.e. goals) can be expressed.

Another solution is forced interactive negotiation. In this simple situation, two conflicting policy authors must be put in contact in order to negotiate and find a resolution between their conflicting policies. Intuitively, this is not a good solution if the end user wishes to have an automated process.

**ECAs** ECA rules can also specify goal constraints or functional rules. In either case, ECA rules need triggers. We have identified, through the mapping of services to tasks, the following events that are applicable:

**Workflow entry/completion/failure:** Policies may be applied at the workflow level (including sets of workflows). This level includes the commencement of a workflow, its successful completion and an abnormal completion with no compensation, i.e. an error result.

**Task entry/completion/failure:** Similar to the workflow level but this time based on tasks. A task failure does not imply a workflow failure, but instead a choice of control flow outputs from the task.

**Service entry/completion/failure:** Again similar to the previous, but based on services. A service failure does not imply a failed task as a policy here can recover the task processing. Conversely, a service success can theoretically lead to task failure.

It is our opinion that these are the most relevant and interesting triggers in a workflow from a control-flow perspective. A service is a black box, thus we cannot see inside it to recognise any triggers. Conversely, the workflow is the highest level at which we can inspect the system, since all policies can be applied no higher than this. We do, however, recognise that there may be further triggers available, especially if one considers data, constraints and resources, which are out of the scope of this paper.

To demonstrate the use of trigger points and error handling (with policies) in a long running transaction, we use simple example as follows:

*Example 2.* Consider a workflow to make a drink and then consume it, plus a separate workflow to purchase coffee granules (Fig. 4). The workflow is augmented with policies that state:

*"if it is morning, I would like a coffee. Otherwise I would like tea"*;
*"if there is no coffee, I would like tea"*; and
*"if there is no coffee or tea, buy some more coffee granules"*.

The time of day is thus important to the final objective of the initial task (*makeCoffee* or *makeTea*), but of small significance in this example (we include it to make a point about time being a factor in business processes).

If it is morning, we will try *makeCoffee*. If this fails, we will try *makeTea*. If this succeeds, then the task completes successfully. If not, then we execute the extra workflow to purchase coffee granules. If this completes successfully, we can go back and try *makeCoffee*, which will hopefully work now. Otherwise, should this extra workflow fail, then the main task *makeDrink* has not been compensated and the task ends in an error state.

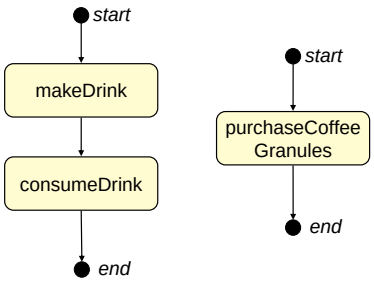


**Fig. 4.** Workflow for making and consuming a drink.

*ECA Conflicts* arise when an event triggers two incompatible policies (i.e. a functional policy conflict, or combined functional/goal conflicts). A functional conflict is one where at least two paths of execution exist, but only one can be chosen. In a state-based system such as a workflow, it exists when the current state is X and two transitions are implied by policies ($P_1 : X \xrightarrow{a} Y$ and $P_2 : X \xrightarrow{b} Z$). This is an example of a shared trigger interaction.

At design time, this can be detected if the policy triggers are not dependent on runtime information. Otherwise, an online detection and resolution method is required. This may include priority sequences as an offline solution or user interaction as an online solution.

Missed trigger interactions occur when a policy forces a workflow reconfiguration, and this avoids the desirable effects of another policy. For example, the cancelling of a doctors appointment may also inadvertently cancel the task of picking up a prescription, since the journey to the surgery is not made.

Sequential action interactions occur when one policy triggers another. For example, we define a simple fail() function that declares a task to have completed abnormally. By calling this function, we might trigger any policies that exist, whose trigger is the current task's failure, even if this is not what was desired (e.g. the failure policy may try to compensate but we might not want that if we have explicitly declared the task to have failed).

Looping interactions occur when one policy triggers another, which triggers the first, etc. Again, provided that the policies are not based on runtime information, these can be avoided at design time. Otherwise, it is difficult to detect and resolve any loops, especially if runtime information is due to change.

### 4.3 Service Selection

Each task inside a workflow has a functional requirement description. In addition it has a default policy. This policy is represented as follows:

```
appliesTo taskId
     when task_entry
       do req(main, Inv, SLA )
```

The function *req* takes three parameters: *main* is the functional requirement of the task, *Inv* is the service invocation parameters and *SLA* is a set of Service Level Agreement (SLA) dimensions. It essentially says that when a task is reached in the control flow, it should execute according to the stated requirements (including finding, binding to and invoking a service).

The primary basis of service selection is the functional requirement. The secondary basis is SLA dimensions. In this set, the policy author can add various non-functional requirements of the service, provided they are measurable in some meaningful way. For example, consider a task *makeCoffee* with a particular requirement that the served cup should be warm. Then, the policy would refine service selection as follows:

```
appliesTo makeCoffee
     when task_entry
       do req(main, Inv, [cupTemperature=''warm'']3)
```

Furthermore, since policies can be added incrementally, APPEL includes composition operators such that policies can be added at runtime[4]. Therefore, many policies stating many different SLA dimensions can be added to even a single task. This is also a method for adding general SLA dimensions across workflows.

*SLA Conflicts* are easily identifiable conflicts. If two or more SLA dimensions address the same service attribute and require different values, then a conflict may exist. These conflicts can be resolved by prioritisation of policies (perhaps the most specific policy first), or by the addition of policy strength indicators. Even then, with two policies conflicting and being as strong as each other, there is still a conflict and a need for resolution.

[3] we expect some knowledge of the service through ontology

[4] policy composition algorithms are not used as they are design time only solutions

Brokerage services can lead to a feature interaction problem, under the auspices of an SLA conflict. For example, suppose a user does not wish to use service X. Instead at runtime, service Y is found, bound to and invoked. However, Y is a broker service and delegates its task to X, and returning X's results to the user. The user is unaware of the involvement of X despite their requirement against using X and thus a feature interaction has occurred. This situation is synonymous with the traditional telecoms example of a feature interaction between Call Forwarding and Call Barring features. The most direct route to resolution in this case is to specify further SLA constraints that require a service to not be a broker, or to provide assurances that the SLA requirements are passed down the brokerage chain.

## 5 Related Work

There has been extensive literature published about policies. They are gaining increasing recognition from implementers as a tool for creating system variability. In addition, there is extensive literature on workflows. Whilst the business processes we discuss can be described in policies or workflows separately, the former method demands too much variable specification and the latter too much static specification.

*Feature Interactions* have, to our knowledge, yet to be reported in the domain of business processes. Weiss et al. [38–41] have reported on feature interactions in Web Services, but in general the subject is confined to telecoms and other network systems. A more formal analysis of feature interaction in processes may lead to specification in a temporal logic in order to provide analysis such as in [6]. In [21], the authors do apply feature interaction to workflows. However their approach does not take into account the instance-based changes via refinement and reconfiguration that we have considered here.

*Workflow Specification.* Apart from natural English, structured languages are often used for expressing processes. BPEL [24] is considered the de facto standard for SOA-based business processes, despite its initial purpose as a service composition language.

More traditional workflow languages are more appropriate for modelling processes. YAWL [33] is a powerful workflow language with semantics based on Petri-Nets. There are alternatives, include SMAWL [31] and others. These solutions may be considered better in terms of describing processes since they abstract away composition details that would be included in those solutions previously discussed. However, they are unable to define high-level requirements for activities or events that occur in the workflow.

A sister approach to the code-based approaches, process calculi and Petri nets offer a formal method in which to express workflows as processes. The formalisms provide operational semantics allowing for reasoning about the process as used in e.g. [15] and [9].

The most widely-accepted universal process notation for business processes is the Business Process Modelling Notation [13] (BPMN). This graphical notation also describes process flows, though somewhat more structured through the use of swimlanes to describe different roles in the process. One particular advantage of BPMN is that it can be used to model a BPEL process [42]. However, BPMN is still limited by its inability to express service selection criteria including non-functional service properties [25].

*Workflow Adaptation* is normally viewed at the overall workflow level. Despite the reported need for flexibility in executing workflows, this is generally achieved through some process reengineering, such as in [18]. Workflow Patterns [17], are a common tool for expressing frequently-occurring patterns in workflows, do allow a certain degree of adaptation. Of particular interest are the insert case and delete case patterns. We consider workflow patterns as relevant work, but the differences exist between their offline design nature, and our online approach to analysis of feature interactions and workflow configuration.

*Policies* are descriptive and essentially provide information that is used to adapt the behaviour of a system. Most work deals with declarative policies. Examples are the formalisms to define access control policies, and to detect conflicts [30, 14]; formalisms for modelling the more general notion for usage control [43]; formalisms for Service Level Agreement, i.e. to specify client requirements and service guarantees, and to *sign* a contract with an agreement between them [22].

RuleML (www.ruleml.org) is a language for rule-based and knowledge-based systems, and allows Web-based rule storage, retrieval and interchange. Like APPEL, it is XML-based and allows for the definition of ECA rules (note that for readability we have not used APPEL's XML syntax in this paper). These rules can be translated through XSL transformations, depending on the application being used.

None of this has been linked to workflows; there has been an initial discussion on linking policies with workflows, presenting the fundamental ideas in [11, 10].

*Workflows and Policies* are combined by Wang [37] in the Policy-driven Process Design (PPD) methodology. Policies are linked to workflows by extracting processes from real business policies and using a common logic to unify them. However the work is more focussed towards extracting new process, rather than affecting current ones. Though Wang does mention insertion and deletion, with respect to control flows, it is only in an overview of the effects on all aspects of workflows (including constraints, data and resources). Furthermore, Wang makes no use of Service Oriented Architecture.

Verlaenen et al [34] have a similar approach, in that policies are used to change workflows. However, the authors use a weaver and policy composition algorithm, indicating an offline approach. Our work specifically addresses the online state.

## 6 Summary and Further Work

In this paper we considered interactions in the context of Service oriented Computing – in particular we considered systems that are described by a workflow that is subject to a number of policies capturing variability. The tasks in the workflow are implemented by services. The two main contributions are a description of the problem domain highlighting and a classification of conflicts in that domain. With respect to the former we identified differences in two major aspects with respect to traditional FI settings: the role of the base system and the longevity of effects of policies. With regard to the latter we presented three classes of interactions: one between policies (policy conflict), one between policies and workflows and one dependent on service selection.

Future work includes the formalisation of StPowla, that is the development of a formal semantics for the workflow part which will allow to extend the conflict reasoning techniques for Appel to be extended to the interaction of policies and workflow.

## Acknowledgements

This work has been partially sponsored by the project SENSORIA, IST-2005-016004.

## References

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2003.
2. Daniel Amyot and Luigi Logrippo, editors. *Feature Interactions in Telecommunications and Software Systems VII, June 11-13, 2003, Ottawa, Canada*. IOS Press, 2003.
3. Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. Theoretical foundations for compensations in flow composition languages. In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 209–220. ACM, 2005.
4. Muffy Calder, Mario Kolberg, Evan H. Magill, Dave Marples, and Stephan Reiff-Marganiec. Hybrid solutions to the feature interaction problem. In Amyot and Logrippo [2], pages 295–312.
5. Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, 2003.
6. Muffy Calder and Alice Miller. Using spin for feature interaction analysis - a case study. In Matthew B. Dwyer, editor, *SPIN*, volume 2057 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2001.
7. Marlon Dumas and Arthur H. M. ter Hofstede. Uml activity diagrams as a workflow specification language. In Martin Gogolla and Cris Kobryn, editors, *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2001.

8. José Luiz Fiadeiro, Antónia Lopes, and Laura Bocchi. A formal approach to service component architecture. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 193–213. Springer, 2006.
9. Xiang Fu, Tevfik Bultan, and Jianwen Su. Formal verification of e-services and workflows. In C. Bussler, R. Hull, S. A. McIlraith, M. E. Orlowska, B. Pernici, and J. Yang, editors, *WES*, volume 2512 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2002.
10. Stephen Gorton and Stephan Reiff-Marganiec. Policy support for business-oriented web service management. In J. Alfredo Sánchez, editor, *LA-WEB*, pages 199–202. IEEE Computer Society, 2006.
11. Stephen Gorton and Stephan Reiff-Marganiec. Towards a task-oriented, policy-driven business requirements specification for web services. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 465–470. Springer, 2006.
12. Stephen Gorton and Stephan Reiff-Marganiec. Policy driven business management over web services. In Rodosek and Aschenbrenner [29], pages 721–724. Poster proceedings.
13. Object Management Group. Business Process Modelling Notation (BPMN) specification. http://www.bpmn.org, Feb 2006.
14. Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *CSFW*, pages 187–201. IEEE Computer Society, 2003.
15. Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In Klaus-Dieter Schewe and Xiaofang Zhou, editors, *ADC*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003.
16. IBM. Service component architecture. http://www.ibm.com/developerworks/library/specification/ws-sca/, 2007. Last accessed 4 June 2007.
17. Workflow Patterns Initiative. Workflow patterns, 2007. accessed 24 July 2007.
18. Beat Liver, Jeannette Braun, Beatrix Rentsch, and Peter Roth. Developing flexible service portals. In *CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 570–573, Washington, DC, USA, 2005. IEEE Computer Society.
19. Emil Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. Software Eng.*, 25(6):852–869, 1999.
20. Carlo Montangero, Stephan Reiff-Marganiec, and Laura Semini. Logic-based detection of conflicts in APPEL policies. In *FSEN2007*, Lecture Notes in Computer Science LNCS. Springer Verlag, 2007.
21. Y. C. Ngeow, D. Chieng, A. K. Mustapha, E. Goh, and H. K. Low. Web-based device workflow management engine. In *MUE*, pages 914–919. IEEE Computer Society, 2007.
22. Rocco De Nicola, Marzia Buscemi, Laura Ferrari, Fabio Gadducci, Ivan Lanese, Roberto Lucchi, Ugo Montanari, and Emilio Tuosto. Process calculi and coordination languages with costs, priority and probability. 2006. SENSORIA Technical Report.
23. OASIS. UDDI: Universal Description, Discovery and Integration. http://www.uddi.org, 2007. Last accessed 4 June 2007.
24. OASIS. Web services business process execution language. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007. Last accessed 4 June 2007.

25. J. O'Sullivan, D. Edmond, and A. H. M. ter Hofstede. Formal description of non-functional service properties. Technical Report FIT-TR-2005-01, Queensland University of Technology, Brisbane, Feb 2005.
26. Nathaniel Palmer. BPM & SOA. 2005. http://aiim.org/article-docrep.asp?ID=30562, last accessed 4 June 2007.
27. Stephan Reiff-Marganiec and Kenneth J. Turner. Feature interaction in policies. *Computer Networks*, 45(5):569–584, 2004.
28. Stephan Reiff-Marganiec, Kenneth J. Turner, and Lynne Blair. APPEL: the ACCENT project policy environment/language. Technical Report TR-161, University of Stirling, 2005.
29. Gabi Dreo Rodosek and Edgar Aschenbrenner, editors. *IM2007: 10th IFIP/IEEE Symposium on Integrated Network Management*. IEEE, 2007.
30. François Siewe, Antonio Cau, and Hussein Zedan. A compositional framework for access control policies enforcement. In Michael Backes and David A. Basin, editors, *FMSE*, pages 32–42. ACM, 2003.
31. Christian Stefansen. Smawl: A small workflow language based on ccs. In Orlando Belo, Johann Eder, João Falcão e Cunha, and Oscar Pastor, editors, *CAiSE Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
32. UN/CEFACT and OASIS. Electronic business using extensible markup language. http://www.ebxml.org/, 2007. Last accessed 4 June 2007.
33. Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. Yawl: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.
34. Kris Verlaenen, Bart De Win, and Wouter Joosen. Towards simplified specification of policies in different domains. In Rodosek and Aschenbrenner [29].
35. W3C. SOAP. http://www.w3.org/TR/soap12-part1/, 2007. Last accessed 4 June 2007.
36. W3C. WSDL: Web Service Description Language v2.0. http://www.w3.org/TR/wsdl20/, 2007. Last accessed 4 June 2007.
37. Harry Jiannan Wang. *A Logic-based Methodology for Business Process Analysis and Design: Linking Business Policies to Workflow Models*. PhD thesis, University of Arizona, 2006.
38. Michael Weiss. Feature interactions in web services. In Amyot and Logrippo [2], pages 149–158.
39. Michael Weiss and Babak Esfandiari. On feature interactions among web services. In *ICWS*, pages 88–95. IEEE Computer Society, 2004.
40. Michael Weiss, Babak Esfandiari, and Yun Luo. Towards a classification of web service feature interactions. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2005.
41. Michael Weiss, Babak Esfandiari, and Yun Luo. Towards a classification of web service feature interactions. *Computer Networks*, 51(2):359–381, 2007.
42. Stephen. A. White. Using BPMN to model a BPEL process. *BPTrends*, 2005. http://www.bptrends.com, accessed on 15/03/06.
43. Xinwen Zhang, Francesco Parisi-Presicce, Ravi S. Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.