EVOLUTION OF SERVICE PROCESSES BY RULE BASED TRANSFORMATION

Christian Zirpins[1] and Giacomo Piccinelli[2]
[1]*University of Hamburg, Germany,* [2]*University College London, UK*

Abstract: The notion of service is closely coupled with the notion of process in general and of workflow in particular. Processes capture the coordination logic for the various resources involved in the realisation of the service content. Moreover, processes drive the actual delivery of a service. Internal processes underpin the capabilities of a service provider. Delivery processes underpin contractual agreements between service providers and consumers. In both cases, the ability to adapt service processes in response to changing environmental conditions is fundamental. Change must be rapid but at the same time accurate and consistent. In this paper, we present the framework for automated process transformation developed within the context of the FRESCO (Foundational Research on Service Composition) initiative. The conceptual part of the framework builds on the standard workflow meta-model proposed by the WfMC (Workflow Management Coalition). The change logic is expressed by transformation rules that can be automatically applied to the processes underpinning a service. The technical part of the framework specifically targets Web service platforms and BPEL (Business Process Execution Language).

Key words: Inter-Organisational Integration, Cooperative Interaction Processes, Electronic Services, Process Evolution, Rule-Based Workflow Transformation

1. INTRODUCTION

Processes in general and workflow in particular are at the core of services and service-oriented computing (Papazoglou and Georgakopoulos, 2003). The realisation of a service depends on the value-added coordination of multiple resources. The coordination logic must be expressed in a way that is at

the same time easy to understand for the human designer and effectively manageable by the execution infrastructure. Workflow frameworks of the type proposed by the Workflow Management Coalition (WfMC, 2004) include process models and notations, organisational and resource models, integration and execution models, development methodologies, as well as execution and management platforms.

Services in general and families of services in particular depend on a multiplicity of processes. A first categorisation of service processes is based on the distinction between internal and delivery processes. Internal processes encompass all the activities that service providers perform in order to realise the core capabilities of a service. Using the freight environment as an example, the core capabilities of a freight service revolve around the handling and transportation of goods. Delivery processes encompass all the activities that service providers perform in order for the service consumer to access and use the core capabilities of a service. In the freight example, this could be advertisement of destinations and spare capacity, contract negotiation, customer interaction or notification management. Service processes are tightly coupled, both within and across categories. As an example, the possibility to deliver progress reports to customers requires specific action (e.g. scanning) to be introduced in the design of the goods-handling processes. Changes at one level must be consistently propagated at all levels. Failing to do so may result in operational inefficiencies as well as potential breach of contractual agreements. The ability to manage change is vital for service providers.

The observation at the base of our work is that the structural change required for a process often reflects a change in overall logic of the service. For example, the scanning steps introduced in a goods-handling process probably derive from the intention of a service provider to include notification facilities in a freight service. The immediate problems with a direct change approach are time and precision. Manual analysis and change of all the processes related to a service is time consuming and error prone. Moreover, the lack of a systematic approach to process adaptation makes organic process evolution extremely difficult. The approach we propose is to capture formally the adaptation requirements for service processes in the form of pattern-based transformation rules. The approach is substantiated by a rule-definition language and related process transformation tool.

In Section 2, we outline the relations between services and processes. In section 3, we discuss the need and scope for automatic process adaptation, and we detail our proposal for a rule-based approach to change. Sections 4 and 5 provide a description of the actual rule language and process transformation tool included in the FRESCO toolkit (Zirpins et al., 2004). Related works are discussed in Section 6. Conclusions from our initial experience are reported in Section 7.

2. PROCESS BASED SERVICES

Effective and efficient support of organisational services depends on the ability to organise and structure the variety of their internal and external processes as well as the mutual interrelations between them. A consequent service model has to consider both, a conceptual part that reflects the background of organisational services as well as a precise representation that allows for automated processing.

Our *conceptual service-model* (Piccinelli et al., 2003b) defines a view on services that is provision-oriented and service-centric. There, cooperation procedures that constitute atomic, self-contained parts of a service-relationship are exposed by so-called *capabilities.* In particular, capabilities represent *purpose, interaction logic* and *resulting artefacts* of the cooperation between organisational *roles.* A service is made up by a set of such capabilities.

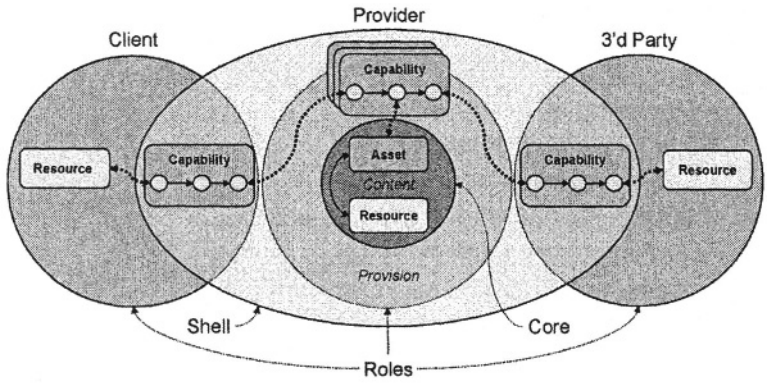

Figure 1. FRESCO service model

A distinctive characteristic of the model is a separation of capabilities in terms of service content and -provision. *Content* reflects the purpose of a service (e.g. moving goods). It is assumed that it arises from specific resources of the provider (e.g. internal processes, knowledge, people, machines, etc.). To represent service content, interactions procedures with such resources are explicitly exposed as meaningful units of content (e.g. transport tracking...) by capabilities referred to as *assets.* Assets don't contain any further cooperative interaction but resource binding (Bussler, 2002) and have to be *provided* to clients indirectly by other capabilities. Assets are grouped into a *service core* representing the complete content. *Provision* addresses procedures that drive a service and make available content (e.g. negotiating terms and conditions, incorporating assets, etc.), whereby control

is exclusively and proactive. Service provision capabilities (hence called "capabilities") are grouped around core assets in a layer called *service shell.* Within a shell, capabilities are mutually interrelated and share a common view on roles and provision-relevant information. Interrelations embody the overall behaviour of provision by defining the global interplay of capabilities. A *service* is fully characterised by defining the basic core and, above all, the enabling shell (fig.1). Our focus is on the latter.

The conceptual service-notion is further substantiated by an architectural model referred to as *service-oriented architecture* (SOA). SOA uses workflow (wf) concepts to define a service as a partitioned set of interrelated components with precise interaction behaviour, where a subset C (service resources) represents interacting participants and a subset S (shell capabilities) represents their cooperation patterns.
In detail, a *role* $r \in R$ is responsible for a set of resource components C_r that are necessary to engage in a service relationship. Actually components are given by their communication endpoints $E_r = \cup e_c, c \in C_r$ that represent service interactions. Within an interaction, data artefacts $D_e \in D$ are communicated. Shell capabilities appear as *glue* between ports, representing a self-contained cooperation task.

The shell is a set of capabilities $S \subset C \times R \times P^*$ where each $s = (c, r, P_s)$ represents a component that is explicitly bound to a role and enforces a set of interaction processes. A process $p \in P_s$ defines a set of transitions $T_p \subset A_p \times A_p$ that forms a precedence graph between interaction activities $A_p \subset E_c \times R \times D^* \times \{in, out\}$. For a capability $s = (c_s, r_s, P_s)$, an activity $a_1 = (e_{cs}, r_s, D_1, in)$ represents an incoming interaction that is externally initiated and includes the communication of artefacts D_1 with endpoint e_{cs} provided by the capability's role r_s itself. An activity $a_2 = (e_x, r_y, D_2, out)$ represents an outgoing interaction where the communication of artefacts D_2 is initiated by r_s and the endpoint e_x is provided by some other role r_y. For two capabilities $s = (c_s, r_s, P_s)$, $t = (c_t, r_t, P_t)$, the interaction processes $p_1 \in P_s$, $p_2 \in P_t$ are *composed* by two activities $a_1 = (e_{ct}, r_t, D, out)$, $a_2 = (e_{ct}, r_t, D, in)$ where a_1 defines an outgoing interaction of p_1 and a_2 defines an incoming interaction of p_2.

The capability-notion from the conceptual model maps to SOA in the sense that implicit semantics of architectural elements (e.g. by ontology-associations) define the *purpose* of *interaction logic* that emerges from the flow of interaction activities and *results* in the flow of data artefacts. Furthermore, the concept of capability interrelation is achieved by composition of interaction processes.

As the SOA model is based on fundamental workflow concepts, it can be mapped to the WfMC metamodel (WfMC, 2002). Subsequently, service

specifications, referred to as *service-schemata,* are represented in the XPDL workflow language.

For illustration, we will outline the example of a *compound transport service.* This logistic end-to-end service combines the transport of goods over multiple legs that are served by different carriers. Related organisational roles include FreightMixer (F) the compound service provider, various transport carriers (T_x, where x is the leg number they serve), an insurer (I) and a customer (C). A major task of F that we will look at, is the control of cooperative handover procedures between T_x and T_{x+1} serving two consecutive legs L_x and L_{x+1}. Handover control includes a standard procedure for trouble free cases as well as a procedure to resolve problems like delay or disaster happened in a leg (fig.2). The overall task is considered as a self-contained capability of the service.

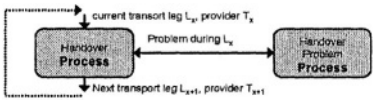

Figure 2. A compound end-to-end transport service

In more detail, the standard handover procedure between a current leg L_x and the next leg L_{x+1} starts with F waiting for a notification from T_x. In the normal case, F notifies I and C about the partial result as well as T_{x+1} about the beginning of his leg and initiates the next handover procedure (fig.3).

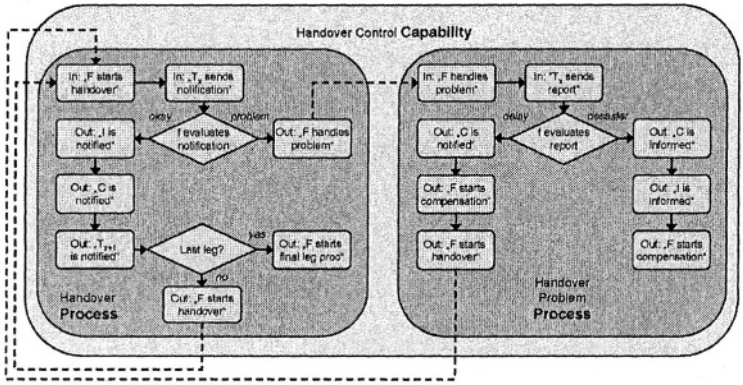

Figure 3. Handover control capability

In case of a problem, F starts a problem handling procedure that is modelled as a separate interaction process. There, F waits for a problem report from T_x. If there was a delay, F notifies C, starts some compensation and

proceeds. In case of disaster, F informs C and I, starts some compensation and stops. The complete set of handover procedures forms the handover control capability. The capability is part of the transport-service shell and associated with F. All activities represent an interaction with a specific component provided by a specific role. In the figure, only (recursive) capability interrelations are shown (dotted lines). The other activities associated with F represent interactions with assets; all others embody interactions with participant resources.

3. SERVICE PROCESS EVOLUTION

Back on conceptual level, services are often required to adapt to change in terms of both the user needs and the operational conditions of service providers. To an extent, flexibility can be built into the structure of a service. An example is the ability of a freight service to support different types of packaging. Still, there are degrees of flexibility as well as operational capabilities that cannot be supported by a given realisation for a service. In the previous example, the possibility to transport perishable goods might simply not be available. Flexibility comes at a cost.

The engineering of a service is substantially based on a complex work of balancing conflicting requirements. In terms of service offer, a rich set of options is likely to attract a wider range of customers, and to provide a better fit for the needs of individual customers. The issues are complexity and cost. More options imply a more complex service design, and a more expensive service infrastructure. In the freight example, the option to transport perishable goods is likely to involve a completely different technical and normative infrastructure. The common approach is to target specific customer segments, and to prioritise the requirements coming from such customers. Still, the more a service is successful the more the need for change tends to emerge. Existing customers will demand better integration and customisation. Prospective customers will demand extensions to the basic service offer. A systematic approach to change is essential for the organic evolution of a service.

The evolution of a service essentially depends on two factors: processes and resources. New capabilities may require new types of resources. In the previous example, the transport of perishable goods requires refrigerated containers. In addition, the realisation of new capabilities requires new processes, for the coordination of new and existing resources. In the example, specific activities will be required for the setting and verification of the temperature for the containers, as well as for the hand-over of the containers at different stages of the transportation. Entirely new processes are occasion-

ally required, but service extension and customisation are mainly based on the adaptation of existing processes. A coherent and consistent evolution of all the processes underpinning a service is essential for the service itself to be operationally efficient and meet customer expectations.

The approach we propose for the evolution of service processes is based on the concept of *transformation rule.* The change logic is captured explicitly in the form of sets of rules, which can then be applied systematically to sets of service processes. A software tool supports the automatic application of the rules to the processes. The formal specification of the rules, a description of the software tool, and an application example are presented in the following sections.

The current model for the transformation rules is based on pattern-matching and direct replacement. The service designer specifies patterns for the portions of a process that need modifications, as well as the changes to apply to the actual processes that match the pattern. The simplicity of the model reflects fundamental requirements such as usability and precision. Current practices revolve around direct changes to process specifications based on a find and replace approach. The rule-based solution we propose builds on current practices in an attempt to improve adoption. Techniques that are more complex would involve costly learning efforts for service designers. Most importantly, the simplicity of the model makes it easier for service designers to appreciate and verify the impact of change. Service designers are ultimately responsible for the result of changes and modifications. Building confidence in the change model is essential.

A direct benefit of explicit formalisation for the changes required to the service processes is consistency. A complete view of the change logic facilitates a systemic view of the impact that the changes will have on the service. Relations between different types of changes become more visible, and conflicts as well as synergies become apparent. A coherent view of the change plan enables a more direct validation of process changes with respect to the actual service evolution requirements. In particular, an explicit evolution plan provides a base for tracing service-level changes to structural changes in service processes, and ultimately to service realisation and delivery.

4. RULE BASED SERVICE TRANSFORMATION

In the SOA model, the definition of service processes is based on workflow concepts. The subsequent approach to systematic change of SOA service-schemata applies a rule based *workflow transformation language* that is described in this section. This language allows describing change strategies for general workflow configurations that can be globally enforced. Thereby

the effects of change can be individually restricted to a useful context (e.g. only service processes for private-customers are to be changed but not those of business-customers). Within this context, processes containing an inappropriate configuration are identified. Changes are given as transformation instructions from the base-configuration into a desired target-configuration. These transformations are then applied to all selected processes in a complete, precise and reversible way.

Change strategies are structured into sets of individual transformation rules. Each rule contains a *process-selection-part* to restrict the application context, a *match-part* to identify the base-configuration within processes of the context and a *replacement-part* to transform the processes as desired.

Systematic change is fundamentally based on *workflow patterns* that allow reliably identifying specific structures throughout an extensive collection of processes. Generally, a pattern enumerates and names workflow elements with respect to a set of match-conditions:

```
<pattern type>
  <match condition1>,
    …
  <match condition n>;
```

Each of the match-conditions describes requirements for a subset of workflow elements. Workflow elements that satisfy the requirements are bound to a specific name. Requirements are given in terms of type, cardinality and state:

```
<element type> {<cardinality>} : <binding name>
  <Boolean expression>;
```

Type requirements restrict elements to a specific type. Thereby, types are classes of workflow elements as specified in the XML Schema of XPDL (WfMC, 2002).

Cardinality requirements restrict a valid match to a specific number of elements. By default, exactly one element is matched. Other ranges can be specified by different modifiers. The *optional* (?) modifier makes a match condition non-obligatory. An *asterisk* (*) causes multiple bindings (including zero) of elements and results in a set that contains all matching elements except for those matched in previous parts of the pattern. *Existence* (+) is similar to asterisk, but requires the element set to be non-empty. *Exclusion* (−) acts as a guard because pattern matching fails in case any such element exists. As a special case of condition, exclusion can be defined for a conjunctive combination of sub-conditions.

State requirements are specified as a Boolean expression in terms of element attributes and associations. Within expressions, attribute values of all elements bound within the whole pattern can be accessed, evaluated and

combined by various operators. In addition to standard element comparison (==, !=), numerical comparison (<, >, <=, >=), arithmetic (+, -, *, /) unary (+, -) and Boolean (`and`, `or`, `not`) operators, also string operations (`contains`, +) ant type castings (`str`, `int`, `bool`, `date`) are possible. *Contained elements* (specified by XPDL as parts of an XML Schema complex type) are either accessible via dot notation or by their obligatory Id attribute (e.g. `myProcess.activities["start"].name`).

In Transformation rules, the *process-selection-part* is given as a specific *selector-pattern* that consists of exactly one *selector-condition.* The *selector-condition* always prescribes the element-type `Process` together with the + modifier. The state-requirement allows specifying an expression that describes the process instances to be considered. In the following example, all processes are pre-selected, whose name attribute contains "handover" and who are part of a workflow package, created before 2004. Also, note that the expression includes type conversion.

```
RULE "transport: conditionally add insurer notification"
FOR_ALL_PROCESSES p
  p.name contains "Handover",
  date(p.package.packageHeader.created)
  < date("01/01/2004 00:00");
```

The *match-part* of a rule is specified by a *match-pattern.* The match-pattern prescribes a specific configuration of arbitrary workflow elements whose existence is a precondition for further processing. Additionally, it declares the binding of element enumerations that result from condition matches, referred to as *condition-match-enumerations,* for further processing. The complete match of a pattern results in a set of condition-match-enumerations referred to as *pattern-match-set.* If a condition is not unique, it can be possibly matched in multiple ways. Subsequently, a match-set is one permutation of the possible condition-match-enumerations. Here, two application semantics are possible: Either to apply the remaining rule for all match-sets individually *(match all)* or to apply it exactly once *(match once).*

The following example illustrates a pattern with match-all semantic and exclusion of two simultaneous conditions. It describes a specific single activity "compensation" that is only matched if there does not exist another specific activity "information" as well as a transition that links them together. If this configuration is found in a process multiple times, the rest of the rule will be applied for all occurrences of "compensation" activities individually.

```
MATCH_ALL
  Activity start_compensation:
    start_compensation.name == "F starts compensation";
  EXCLUSION "excludeIfInformed"
    Activity inform_insurer:
      inform_insurer.name == "I is informed";
```

```
Transition inform_compensate:
  inform_compensate.from == inform_insurer,
  inform_compensate.to == start_compensation;;
```

A rule is concluded by a *replacement-part* that defines how to transform the match-set. It consists of transformation operations for all condition-match-enumerations of a match-set. The default operation is removal so that an empty replacement-part leads to the removal of the complete match-set. To preserve a condition-match-enumeration it has to be explicitly referenced in the replacement part whereby all element attributes can be modified. Additionally, the creation of new elements can be instructed that are to be added to the process. After creation, attributes are initialised with default values that can be modified as appropriate. Syntactically, element creation consists of a type, a name and various attribute value assignments.

```
<element type> : <binding name>
  <attribute name> = <expression>,
    …
  <attribute name> = <expression>;
```

To affect the placing of elements within a process, a special attribute called "container" is introduced that allows accessing and changing the parent of any element. When using references to optional elements of the match-set, the transformation operation has to be declared optional, too.

In the example, the match-set contains only a single activity. The exemplary replacement-part preserves this activity by referencing it initially. Additionally it instructs the creation of another activity and a transition that connects both activities.

```
REPLACE_WITH
  start_compensation;
  Activity notify_insurer:
    id = "notify_insurer",
    name = "Notify insurer",
    implementation = No();
  Transition compensation_notify:
    id = "compensation to notify",
    from = start_compensation,
    to = notify_insurer;
```

To enforce changes, specified by a number of rules, the aggregated rule set is applied to a pool of workflow processes as follows:

1. Rules are taken from the rule set one at a time. They are applied to a process pool individually as described in the next steps.
2. Initially, the selector-pattern is applied to the process pool. For each process within the pool, the pre-selection condition is evaluated exactly once. If the Boolean expression evaluates true, the process is selected for further processing.

3. In the next step, each pre-selected process is checked against the match-pattern. For each condition of the pattern, corresponding elements are collected that satisfy the type- and state-requirements. A subset of these elements is bound with respect to the cardinality requirement whereby bound elements are excluded from further matches. In terms of sequence, obligatory conditions (none or + modifier) are matched first, followed by optional single matches (?) and finally optional enumerations (*). In case of non-unique patterns, all permutations of binding combinations are considered. The pattern matching fails if a) a match condition fails or b) an exclusion condition is violated. In this case, the processing is terminated. Otherwise, the resulting matches are structured into pattern-match-sets of condition-match-enumerations, each set representing one binding permutation. Depending on the match-semantic, either all (match-all) or exactly one match-set (match-once) is further processed.
4. Finally, the sequence of match-sets is transformed one at a time as specified in the replacement part of the rule. All non-referenced condition-match-enumerations are removed from the process. All others are modified as instructed. New elements are created and added to the process. For the resulting process, a basic consistency check is done that detects obvious violations of the process structure and rolls back the transformation in case of an error. The transformation also fails in case of unbound references.

To illustrate the usage of transformation rules for changing service processes, we revisit the transport-service example. Let's assume that Freightmixer's management decides to change the business procedure. In a first change, the insurer ought to be informed after a problem compensation procedure was started. For obvious reasons, this is not necessary if the insurer was informed directly before. For this change, the rule introduced before can be used to detect and transform every such situation.

In a second change, all interaction processes of any cooperation ought to be stripped down by removing costly notifications of the customer. This can be achieved by the following rule:

```
RULE "transport: remove notifications"
FOR_ALL_PROCESSES p
  p.name contains "Handover";
MATCH_ONCE
  Activity{*} a IN notificationActivities:
    a.name contains "C is notified";
REPLACE_WITH
  FOR_EACH a IN notificationActivities:
    name = "activity without function",
    route = Route();
```

The second rule is an example of match-once semantic. This is sufficient because the match condition of the pattern is unique. Elimination of notifica-

tions is done by changing interactions into inactive routes, thereby omitting a complicated change of transitions for the sake of simplicity. After adding both rules to a combined rule set, they are applied to the service specification. The result is a change in both interaction processes of the handover control capability (fig. 4). In both processes, the notification activities disappeared. (Actually, they are changed to trivial routing activities that are simplified as single transitions in fig. 4. This is also visible in fig.6, where the associated tool shows an out/in comparison ("diff") of the change effects.) In addition, the insurer is informed after a compensation activity, but only in the case where he was not already informed before.

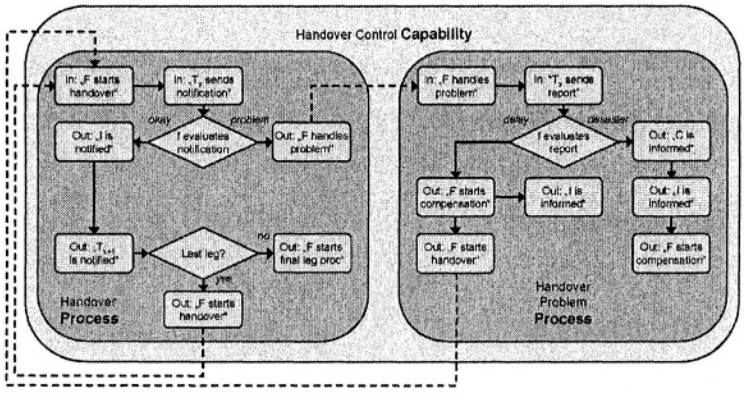

Figure 4. Changed handover control capability after rule application

5. PROTOTYPE

As proof of concept, we developed the functionality of rule based workflow transformation as part of a prototype environment referred to as *FRESCO Toolkit or FrescoTK*[24]. FrescoTK is an implementation of the FRESCO SOA. It implements a fundamental way for service realisation as well as a set of crucial service engineering mechanisms (Zirpins et al., 2004).

Major implementation technologies include the Open Grid Service Architecture OGSA (Foster et al., 2002) together with specific aspects of the more general Webservice Architecture (Tsalgatidou and Pilioura, 2002) and

[24] FrescoTK is available under academic free licence at www.servicecomposition.org

a BPEL enabled workflow management system. Fig 5 provides an architectural overview.

In the toolkit, FRESCO *services* are implemented as sets of OGSA Gridservice-components, divided in two subsets: capabilities and resources. *Resources* are implemented as conventional Gridservices, extended by a set of OGSA service-data attributes. *Capabilities* are components that can be accessed as Gridservices too, but also proactively enforce service specific behaviour.

While resources have to be realised and provided outside the scope of FrescoTK, capabilities are generated and deployed by the FrescoTK. on the basis of a service-schema specification. Internally, they consist of a BPEL enabled workflow engine that drives the capability's interaction processes. As BPEL engines are based on Webservice technology, they are not originally applicable in a grid environment. Thus, capabilities feature an integration-architecture to bridge between Grid- and Webservices. This integration-architecture uses various interceptors (represented as proxies and adapters) that translate calls and convert instance management information.

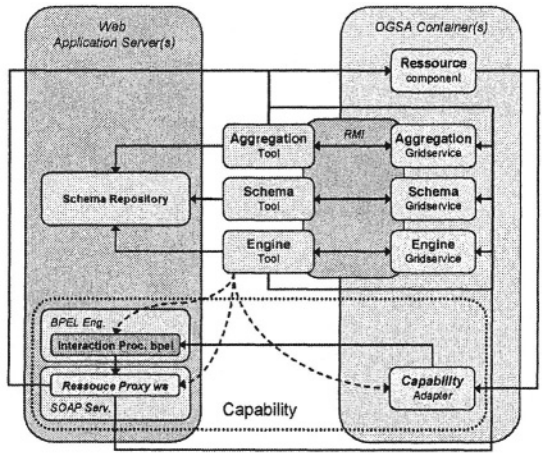

Figure 5. FRESCO Toolkit architecture

The engineering of services is based on a set of *management components.* In brief, these components provide essential functionality to plan, build and run FRESCO services by service-schema management (Schema Tool), service-instance creation and resource management (Aggregation Tool) as well as service-capability generation and control (Engine Tool). Management components are designed as tools that provide a GUI for human users. Additionally, they are designed as system support mechanisms that provide an API for programmatic use. Therefore, the components are

implemented as java standalone applications and major management operations are accessible via Grid. Each component is represented as a Gridservice by a proxy, to which it is connected via Java remote method invocation.

The *Schema Tool* (fig.6) holds generic specifications of various service-schemata and makes them programmatically accessible. Its vital characteristic is the ability to apply a variety of transformations to them that allow for controlled change as well as translation into executable format. Management operations – provided via GUI and Grid – include *storage, retrieval, removal* and *browsing* of service-schemata, *translation* of service-schema into executable BPEL process-schemata as well as *change* of service-schemata by application of customised transformation rules.

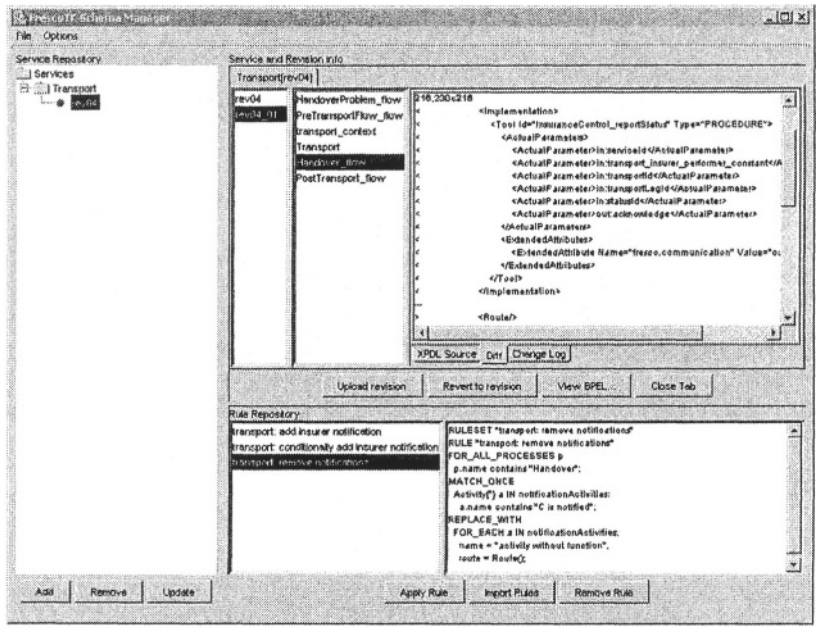

Figure 6. FrescoTK service-schema tool

Transformation rules are held in a *rule repository.* They are directly entered or imported from a file, whereby validation takes place. Then, they can be applied to service-schemata that are selected from the schema repository. Revision management allows tracking changes and doing rollbacks in case. A rule is always applied to one specific revision of a schema. If it leads to changes in any of the revision's packages, a new local revision is created. Stable revisions can be chosen to be persisted in the repository.

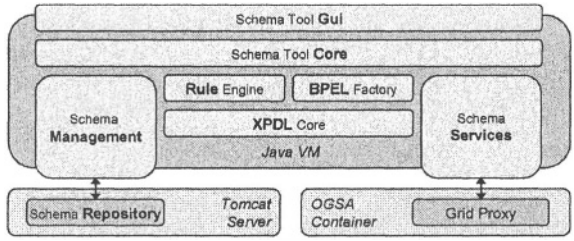

Figure 7. Schema Tool Architecture

Rules are processed by the *rule engine* that is embedded in the schema-tool architecture (fig.7). There, it is used by the *schema-tool core* to provide transformation services for GUI and Grid (via RMI-connected proxy) access. The rule engine is based on the FRESCO XPDL core; a Castor-generated XML binding API (ExoLab, 2004) that implements XPDL elements as beans, using Jakarta Beanutils (Jakarta-Project, 2004). Here, support of mapped and indexed bean properties allows for rich rule expressions; i.e. `process.activities[5]` (indexed property) or `extendedAttributes["myAttribute"]` (mapped property). Such expressions are implemented using ANTLR (Parr, 2004) and a grammar of this language is available as part of the FRESCO Toolkit's source code. The engine itself implements the change algorithm as explained in the previous section.

6. RELATED WORK

The FRESCO service model is mainly related to work from three different areas of inter-organisational process management. In *BPM (business process modelling & management),* virtual enterprises and inter-enterprise-processes are a major concern (Bussler, 2001, Georgakopoulos et al., 1999, Perrin et al., 2003, Schuster et al., 2000, Baïna et al., 2003). The DySCo project (Piccinelli et al., 2003a) introduced a basic process-based service model and developed methods to deduce participant-related sub-processes for service control. *Inter-organisational workflow* adds concepts of distributed control structures (van der Aalst, 1999) and technology of related management systems (Mecella et al., 2001, Colombo et al., 2002). Finally, the focus of *service composition* (Papazoglou and Georgakopoulos, 2003) is on combining atomic functions of loosely coupled systems (Webservices) by processes (Casati et al., 2001, Dumas et al., 2002, Bhiri et al., 2003). The view on services adopted in FRESCO combines concepts from these areas as regards inter-organisational business processes, workflow architecture for their control and service technology for their implementation. It adds a cooperation

model with novel service-oriented concepts (e.g. service capabilities with explicit separation of service content and service provision)

The issue of evolution for process specifications has been object of attention for quite a few years, both from the scientific community and the industry. *BPR (Business Process Re-engineering)* was a major area of activity in the late 1990s' (Hunt, 1996). The results produced cover extensive requirements analysis, as well as methodologies and techniques to address the various technical and organizational issues related to process change (i.e. (Sethi and King, 1998, Piccinelli et al., 2002)). BPR activities have been substantially based on direct intervention on process definitions by teams composed of process engineering and business experts. The rule-based approach we propose would complement the operational model used in BPR environments.

A second important line of activity for process evolution is based on *data mining* techniques applied to the process execution logs generated by workflow management platforms (Agrawal et al., 1998, Srinivasa and Spiliopoulou, 2000). The focus is on the identification of patterns in the execution trace of multiple process instances in order to infer areas of improvement for the specification of the process. Patterns can involve individual processes or span groups of related processes (Cook and Wolf, 1998). The patterns identified by the data mining techniques can be combined with the technical and domain expertise of process designers in order to define the evolution strategy for the processes. The rule-based approach we propose can be used to capture and execute the chance strategy.

In terms of *workflow patterns,* fundamental research has been done in the Petri-net community (van der Aalst et al., 2000). Additionally, concepts to ensure consistency of changes (e.g. *workflow inheritance*) where proposed (van der Aalst and Basten, 2002). Our approach can directly benefit from this foundation and helps transporting it to practice.

7. CONCLUSION

The capability to adapt the processes underpinning a service is essential for service evolution and customisation. The resource base of a service provider changes over time. Similarly, customer requirements change and diversify. The alignment between service logic and the underlying realisation and delivery processes of a service can become an issue for providers, and ultimately also for the consumers.

The rule-based framework we propose for the adaptation of service processes balances usability and precision against semantic and abstraction richness. The pattern-matching model used in the transformation rules is close to

the find-and-replace model normally used by service designers. Rules add the benefit of a systematic approach to the specification as well as the application of the change logic. The combined use of the rule model and the tool developed for rule application reduces the risk of omissions and mistakes intrinsic in manual approaches. In addition, direct improvements can be achieved in terms of the tractability of change.

An important benefit of the approach proposed is the possibility to capture an overall change strategy, and to separate definition and application of such strategy. Future work will concentrate on support for the validation of an overall change strategy with respect to service evolution requirements as well as verification of consistency for the set of rules composing a strategy.

ACKNOWLEDGEMENTS

We would like to thank HP Labs Bristol for support of the FRESCO project. Furthermore, we want to thank the FRESCO team, especially Thomas Plümpe and Henning Brandt, for their help to achieve this work.

REFERENCES

Agrawal, R., Gunaopulos, D. and Leymann, F. (1998) *Mining Process Models from Workflow Logs,* In *Proc. Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain*(Eds, Schek, H. J., Saltor, F., Ramos, I. and Alonso, G.) Springer, pp. 469-483.

Baïna, K., Tata, S. and Benali, K. (2003) *A Model for Process Service Interaction,* In *Business Process Management International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003. Proceedings*(Eds, Aalst, W. M. P. v. d., Hofstede, A. H. M. t. and Weske, M.) Springer, pp. 261 ff.

Bhiri, S., Perrin, O., Gaaloul, W. and Godart, C. (2003) *An Object Oriented Metamodel for Inter-enterprises Cooperative Processes based on Web Services,* In *Proc. Modeling and Developing Process-Centric Virt. Enterprises with Webservices (VIEWS'03), Austin, USA.*

Bussler, C. (2001) *The role of B2B protocols in inter-enterprise process execution,* In *Proc. Technologies for E Services. Second International Workshop, TES 2001.*(Eds, Casati, F., Georgakopoulos, D. and Shan, M. C.) Springer, pp. 16-29.

Bussler, C. (2002) *Behavior abstraction in semantic B2B integration,* In *Conceptual Modeling for New Information Systems Technologies. ER 2001 Workshops. HUMACS, DASWIS, ECOMO, and DAMA. Revised Papers Lecture Notes in Computer Science Vol.2465. 2002*(Eds, Arisawa, H. et al) Springer Verlag, Berlin, Germany, pp. 377-89.

Casati, F., Sayal, M. and Ming Chien Shan (2001) *Developing E-Services for Composing E-Services,* In *Proc. Advanced Information Systems Engineering. 13th International Conference, (CAiSE 2001)*(Eds, Dittrich, K. R. et al) Springer, pp. 171-86.

Colombo, E., Francalanci, C. and Pernici, B. (2002) *Modeling Coordination and Control in Cross-Organizational Workflows,* In *Proc. CoopIS/DOA/ODBASE 2002*(Eds, Meersmann, R. and Tari, Z.) Springer, pp. 91 ff.

Cook, J. E. and Wolf, A. L. (1998) *Discovering models of software processes from event-based data. ACM Transactions on Software Engineering and Methodology,* **7**.

Dumas, M., Benatallah, B. and Maamar, Z. (2002) *Definition and Execution of Composite Web Services: The SELF-SERV Project, Data Engineering Bulletin,* **25**.

ExoLab (2004) *Castor Project,* http://castor.exolab.org/, 1.2.2004

Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002) *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,* Open Grid Service Infrastructure WG, Global Grid Forum

Georgakopoulos, D., Schuster, H., Cichocki, A. and Baker, D. (1999) *Managing process and service fusion in virtual enterprises, Information Systems,* **24**, 429-56.

Hunt, V. D. (1996) *Process Mapping: How to Reengineer Your Business Processes,* John Wiley & Sons.

Jakarta-Project (2004) *JaKarta Commons,* http://jakarta.apache.org/commons/, 1.2.2004

Mecella, M., Pernici, B., Rossi, M. and Testi, A. (2001) *A Repository of Workflow Components for Cooperative e-Applications,* In *Proceedings of the 1st IFIP TC8 Working Conference on E-Commerce/E-Business (Salzburg, Austria, 2001)*BICE Press, pp. 73-92.

Papazoglou, M. P. and Georgakopoulos, D. (2003) *Service-oriented computing: Introduction, Communications of the ACM,* **46,** 24-28.

Parr, T. (2004) *ANTLR Translator Generator,* http://www.antlr.org/, 1.2.2004

Perrin, O., Wynen, F., Bitcheva, J. and Godart, C. (2003) *A Model to Support Collaborative Work in Virtual Enterprises,* In *Business Process Management International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003. Proceedings*(Eds, Aalst, W. M. P. v. d., Hofstede, A. H. M. t. and Weske, M.) Springer, pp. p. 104 ff.

Piccinelli, G., Emmerich, W., Zirpins, C. and Schütt, K. (2002) *Web Service Interfaces for Inter-Organisational Business Processes: An Infrastructure for Automated Reconciliation,* In *Proc. 6th International Enterprise Distributed Object Computing Conference (EDOC2002), September 17-20 2002, Lausanne, Swizerland*(Ed, Williams, A. D.) IEEE Computer Society, Los Alamos, California, pp. 285-292.

Piccinelli, G., Finkelstein, A. and Williams, S. L. (2003a) *Service-oriented work-flows: the DySCo framework,* In *Proc. 29th Euromicro Conference, Antalya, Turkey.*

Piccinelli, G., Zirpins, C. and Gryce, C. (2003b) *An Architectural Model for Electronic Services,* University College London, University of Hamburg

Schuster, H., Georgakopoulos, D., Cichocki, A. and Baker, D. (2000) *Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes,* In *Proc CAiSE 2000*(Eds, Wangler, B. and Bergman, L.) Springer, pp. 247-263.

Sethi, V. and King, W. (1998) *Organizational Transformation Through Business Process Reengineering.,* Prentice Hall.

Srinivasa, S. and Spiliopoulou, M. (2000) *Discerning Behavioral Properties by Analyzing Transaction Logs,* In *Proc. of the 2000 ACM symposium on Applied computing 2000, Como, Italy*(Eds, Papadopoulos, G. and Omicini, A.) ACM Press, NY, USA, pp. 281 - 282.

Tsalgatidou, A. and Pilioura, T. (2002) *An overview of standards and related technology in Web Services, Distributed and Parallel Databases,* **12,** 135-62.

van der Aalst, W. M. P. (1999) *Process-oriented architectures for electronic commerce and interorganizational workflow, Information Systems,* **24,** 639-71.

van der Aalst, W. M. P., Barros, A. P., Hofstede, A. H. M. t. and Kiepuszewski, B. (2000) *Advanced Workflow Patterns,* In *Cooperative Information Systems, 7th International Conference, CoopIS 2000, Eilat, Israel, September 6-8, 2000, Proceedings,* Vol. 1901 (Eds, Etzion, O. and Scheuermann, P.) Springer, pp. 18-29.

van der Aalst, W. M. P. and Basten, T. (2002) *Inheritance of workflows: an approach to tackling problems related to change, Theoretical Computer Science,* **270,** 125-203.

WfMC (2002) *Workflow Process Definition Interface -- XML Process Definition Language 1.0 Final Draft,* WFMC-TC-1025, Workflow Management Coalition

WfMC (2004) *Workflow Management Coalition,* http://www.wfmc.org, 20.1.2004

Zirpins, C., Lamersdorf, W. and Piccinelli, G. (2004) *A Service Oriented Approach to Interorganisational Cooperation,* In *IFIF International Conference on Digital Communities in a Networked Society: eCommerce, eBusiness, and eGovernment (I3E) 2003, Proceedings*(Eds, Mendes, M., Suomi, R. and Passos, C.) Kluwer Academic Publishers.