Electronics and ICT as enabler for digital industry and optimized supply chain management covering the entire product lifecycle

Project Acronym:

Productive4.0

Grant agreement no: 737459

Deliverable no. and title	**D2.2 – State of the Art for Complex Workflow Generation**	
Work package	WP 2	Environment for Digital Industry
Task	T 2.2	Data Analytics and Handling services
Subtasks involved		
Lead contractor	Infineon Technologies AG Knut Hufeld, mailto:knut.hufeld@infineon.com	
Deliverable responsible	Thales Nederland BV J.B. van Veelen, bernard.vanveelen@nl.thalesgroup.com	
Version number	v1.1	
Date	13/02/2018	
Status	Final	
Dissemination level	Public (PU)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
05D	TNL	J.B. van Veelen	All

Document History

Version	Date	Author name	Reason
v0.1	21.10.2017	J.B. van Veelen	Initial Draft
V1.0	13.02.2018	Mike Holenderski	Changes after review
	15.02.2018	Alfred Hoess	Final editing and submission

Publishable Executive Summary

This deliverable presents an overview of business process orchestration technologies from the perspective of collaboration using workflows. In the context of Productive4.0, workflows are actionable business processes, which are created in collaboration by the various stakeholders, using automated workflow generation services. Also the execution and quality of service monitoring is established collaboratively.

This deliverable presents the baseline principles for workflow generation and management for such collaborative business operations. It discusses workflow representation, collaborative workflow generation as well as the management capabilities (selection, instantiation, execution and quality of service management) required to successfully complete these collaborative operations.

The discussion of baseline principles concludes with a set of functional requirements that candidate technologies need to fulfil for deployment in the Productive4.0 context.

As the feasibility of automated collaborative workflow generation and management is evaluated by a use case from the Productive4.0 project, operational requirements from that use case are elicited as additional requirements. The combination of functional and operational requirements constitute the envelope for selection of technologies to be used.

Instead of providing an exhaustive overview of available technologies, this deliverable limits its investigation to the major contenders in the domain of process orchestration, such as BPEL and YAWL, and, in addition, the technologies familiar to the Productive4.0 project partners, such as Arrowhead and BRAWL. The surveyed candidate technologies are evaluated against the framework of requirements established in the chapters before. The deliverable concludes with a selection of technologies to be deployed in the Productive4.0 use case.

Table of contents

List of figures

List of tables

1. Introduction

1.1 Context

Smarter and more flexible production, a better use of resources, new standards and a changing work environment are just a few aspects associated with Industry 4.0, also known as Digital Industry. The digital transformation associated with Industry 4.0 will affect almost all industries and our everyday lives. Although the term 'digital transformation' has become popular, industry still has some way to go. Linking the real with the digital world takes more than adding software to hardware. To a large extent companies still lack the fundaments. A hands-on approach and practical implementations are needed as well as tailor-made solutions focusing on the main fields of digital automation, supply chain networks and product lifecycle management. Moreover, an optimal failure risk management has to be ensured.
The Productive4.0 tackles the typical challenges, such as:

- self-configurable supply chain processes and automated order-/contract handling;
- innovative features and fail operational concepts including sensors and actuators - all the way from the IoT-component level to complete autonomous systems like robots or intelligent cars;
- finding technologies that are able to integrate and deal with legacy systems;
- The capability that enables industry to deal with different cycle times at final product level as well as module and component level.

To achieve significant improvement in digitalising the European industry by means of electronics and ICT, concerns addressed in this report are: the representation of business processes as actionable workflows, creating workflows automatically and collaboratively, and, finally, instantiating and orchestrating workflows in a distributed collaborative industrial environment. The report investigates aspects of interest in this context, including representation (WF languages), distributed collaborative workflow generation & management and distributed collaborative Quality of Service management.

1.2 Purpose and Scope

The purpose of this document is to provide an overview of the current collaborative workflow generation & management (CWFGM) and Quality of Service Management (QoSM) that could be deployed in the Industrial Internet of Things (IIoT) context of the Productive4.0 project.
To achieve this purpose this report first documents the baseline principles of what workflows are and explores the relevant aspects of system orchestration using workflows. These aspects are workflow generation, workflow instantiation and quality of service management using monitoring and mitigation based on information in workflows. Evidently, in this context workflows are very elaborate information carriers, documenting all aspects necessary to use the workflow as an actionable business process.

Having set the scope for workflows and workflow processing in the Productive4.0 project, the second part of this report elicits and analyses the requirements of a relevant Productive4.0 use case. The combination of the outcomes of this initial exploration of the ambition and context will yield clear criteria for selection and design in the context of Productive4.0.

Investigating the state of the art and applying the criteria found is the topic of the final part of this report. The purpose of this third part is to identify and analyse candidate technology and select a suitable candidate for evaluation of usability in the context of Productive4.0.

This document focuses on workflows and the manipulation and deployment of workflows in an industrial context. Since workflows are information carriers, they need a representation format, a language, more specifically, they need a language that is useful in the context of application: automated orchestration.
This report will therefore make an inventory of the most relevant candidate languages and orchestration technologies, paying attention to the concerns

- How well does the language fit the technology?
- How well does the technology suit the requirements of collaborative orchestration?
- How well does the combination of language and technology suit the requirements of the Productive4.0 project, and more specifically the use case 9.2?

Fortunately, in the Productive4.0project we do not have to start from scratch. This report will heavily build on results of other projects, especially on the results from FP7 BRIDGE project. Our particular interest is to investigate whether the results of the BRIDGE project can be tailored to a Productive4.0 use case; in this context 'tailoring' implies adjusting and extending the results to fit the particular needs of the Productive4.0 T9.2 use case.

1.3 Document overview

In this report we will first explain what we mean when talking about workflows, workflow generation and quality of service management in chapter 2. We will also extract some high level functional requirements on WF generation and management in the context of the Productive4.0 project. In chapter 3 we will investigate and analyse what the requirements are on workflow generation and quality of service management in the context of the Productive4.0 project, focussing on challenges in product lifecycle management (PLM) and supply chain management (SCM). The requirements will pertain to PLM and SCM in general, but to Productive4.0 use-case defined in task 9.2 on Digital Product Footprint in particular. In chapters 4, 4.3 and 0 we will provide an overview the current state of the art in workflow representation, workflow generation and management and quality of service management respectively in the light of the requirements documented in chapter 3. We conclude this report with some recommendations and conclusions regarding the feasibility of technological solutions regarding workflow generation and management in the context of Productive4.0 in chapter 7.

2. Baseline Principles

2.1 Introduction

In this chapter we will discuss the basic principles of workflows, workflow generation & management (WFGM) and quality of service management (QoSM) based on workflows. We will first investigate what a workflow is and what it means in section 2.2. In section 2.3 we will discuss how workflows can be generated and what cases workflow generation is desirable. Section 2.6 discusses how workflows can be instantiated and, finally, section 2.8, clarifies how the quality of service of systems executing workflows can be managed.

The text in this chapter is based on some of the results from the FP7 BRIDGE project; as documented in the Productive4.0 proposal, we will apply these results in an ePLM context. Introducing the workflow principles according to the lines set out in the BRIDGE project, the text in this chapter has been completely re-edited and extended to form a basis for the efforts in Productive4.0. While the domain of application in the BRIDGE project was crisis

response and management, the results have to be adjusted, extended and modified to fit the needs of the Productive4.0 context.

2.2 Workflows

2.2.1 What are workflows?

Based on the work of Taylor and Gantt aiming to find a rational approach to organizing work in manufacturing processes [1, 2], '*workflow*' is a term used to describe a model of a production process[1].
Wikipedia [3] loosely defines workflows as: "*…A workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information. It can be depicted as a sequence of operations, declared as work of a person or group, an organization of staff, or one or more simple or complex mechanisms.*
From a more abstract or higher-level perspective, workflow may be considered a view or representation of real work. The flow being described may refer to a document, service or product that is being transferred from one step to another…"

In this loose definition a number of characterisations stand out:

1. a workflow is orchestrated, implying it refers to a carefully composed set of interdependent operations. The interdependency of operations is constituted by the artefact[2] that is provided by one operation and serves as input for the next operation. This is why a workflow may be represented as a sequence of operations.
   Pointing out the obvious, a workflow has a purpose, is designed and constructed to achieve this purpose, and is subject to some form of control or management to best achieve its purpose.
2. a workflow is a repeatable pattern of activity; this implies that a workflow can be compared to a recipe describing what to do (including ingredients, timing and conditions) to achieve a certain result.
3. a workflow is based on the organization of resources into (operational) processes. A workflow distinguishes the activity or operation from the actor that is executing the activity. As a consequence, 'acting' resources have to be allocated to tasks (i.e. activities or operations to be performed), whereas 'usable' resources (such as materials, fuel, data or information) have to be designated for processing.

As the name 'workflow' suggests, it documents not just the 'flow', but also the 'work'. In other words, a workflow describes transfers as well as transformations in a process. In addition, we will explicitly include the in- and outputs of the transformations, the artefacts being transferred, in the workflow. Consider the simple observe-analyse-plan workflow in Figure 1:

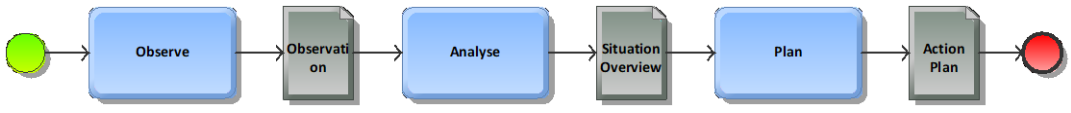


Figure 1: Simple workflow

This workflow is composed of three steps (or '***activities***' used in this document), 'Observe', 'Analyse' and 'Plan'. Each activity produces one output (we will call in-and output

[1] Note that in general a workflow may be used to denote the actual 'execution of activities on the work floor', as well as the abstract 'recipe' to achieve certain results. In this document we will only refer to the abstract recipe using the term workflow and use the term execution for the actual execution of the workflow.
[2] product, information or service

'***artefacts***'), 'Observation, 'Situation Overview' and 'Action Plan' respectively. In this document we will denote **activities** in workflows with blue rounded rectangles, and artefacts with the grey squares-with-a-folded-corner. Arrows denote the exchange artefacts between activities in the workflow. A green circle denotes the starting point of a workflow, the red circle denotes the end point.

A more in-depth discussion about workflows and the underpinning theory can be found in the work of van der Aalst, for example [4].

2.2.2 What are workflows used for?

The concept 'workflow' is closely related to the concept 'business process'. Where a business process is a sequence of activities performed to produce value, a workflow is a sequence of activities performed to produce a certain result or output; thus, a workflow is a process in itself.
Often used as a means of communication, workflows similar to the one presented in Figure 1, can be commonly found on whiteboards, slide shows and drawing pads. In Productive4.0 context we want to push this envelope just a bit further; in literature, it is common to perceive a workflow as an actionable model of a business process, step-by-step describing what to do, to achieve a certain desired result [3.1]. In order to obtain an actionable model of a business process, we need to include into the workflow also relevant aspects of ***quality*** of activities, artefacts and exchange, enabling the orchestration of the workflow from the level of contracting of 'acting resources' (using for example service level agreements. we will call acting resources *actors* from now.), to monitoring of progress and performance, as well as mitigation of deviations in a workflow.

Therefore, in the context of Productive4.0 we want to use the term 'workflow' for a decomposition of a business process into interrelated activities, exchanging artefacts; a workflow is constructed and executed to achieve a certain result (or results), describing all necessary aspects of activities, timing and conditions to achieve the intended result(s).

More specifically, to have a actionable description of a business process, in Productive4.0 context we want workflows to (be able to) document:

1. the **actions**, i.e. what operation is to be executed
2. the **artefacts**, i.e. what the operation is applied to and what it yields. Artefacts are the in- and outputs of operations
3. the **arrows**, meaning the exchange, how artefacts 'flow' from one activity to another.
4. the qualities associated to actions, artefacts or arrows.
    a. qualities of activities
    b. qualities of artefacts
    c. qualities of exchange of artefacts between activities.
5. orchestration information regarding actions, artefacts or arrows, such as allocated resources, service level agreements and progress/performance criteria.

Allocated resources will be documented as **actors**. The other information under item 4. and 5. will be documented in **annotations**. Note that annotations always refer to a specific element in the WF.

Additionally, the approach we want to explore in Productive4.0, involves automated workflow generation & management and QoS management. This means that the specification of configurations of resources in a workflow is not designed beforehand, but generated in response to a set of requirements (called the *need*). This means workflows will have to adhere to some predetermined syntax and semantic.
In résumé, in the context of Productive4.0, a workflow has a number of purposes:

- Description: a workflow **describes** the sequence of activities necessary to achieve a desired output. The desired output is specified in a need (set of requirements) and corresponds to some business goal.
- Valuation: we can **determine** a value for a workflow (given some policy), compare the values of individual workflows (generated in response to the same set of requirements) and **select** the most desirable workflow available.
- Manipulation/Adjustment: generating workflows in an explicit format, enables the manipulation of workflows, in the sense that we can combine (two) workflows into a new workflow[3]. Note that the combination of two workflows yields a (new) workflow, which in turn can be combined with other workflows.
- Execution: we can **instantiate** a workflow by allocating actors to the various activities of a workflow, and configure the actors:
    - instructing the actors where to find their input and where to provide their output,
    - what operation to perform and
    - what quality levels to apply.

  Effectively, the combination of this information constitutes a *contract* or *service agreement* for the actor.
- Monitoring: We can use the contracts originating from instantiation to **monitor** the performance of the instantiated workflow. Of course, detection of deviations should result in some form of **mitigation**.

The bullets "description", "valuation" and "manipulation" are parts of workflow generation and management, whereas "execution" and "monitoring" describe responsibilities for quality of service management.

2.2.3 What do workflows look like?

In the e-PLM systems we envision, workflows are generated in response to a need. The aim is not to produce a graphical representation of a workflow satisfying the need, but to use the workflows as a recipe for system configuration and an agreement on performance; so the graphical form of the workflow will usually be ignored[4]. Instead, since they are generated, the workflows will take the forms of text, containing all the relevant information to instantiate and monitor an organisation of people and machines. As an example, consider the annotated workflow fragment in Figure 2.

[3] Indeed, the principle of workflow generation is iteratively adding pieces of workflow to an existing (partial) workflow.

[4] To create the graphical representations of workflows (WF) in this document, we use ARIS Express [5], a Business Process Modelling tool from IDS Scheer AG [6].

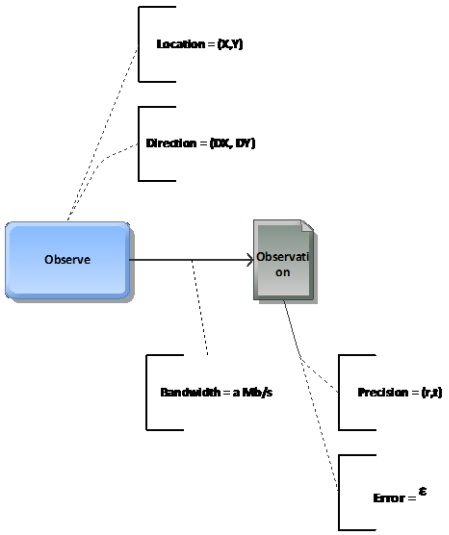


Figure 2: An annotated workflow fragment

The workflow depicted in this example can be put in (pseudo-) code as:

```
Activity (A = Observe () );
Annotation (A, Location = (X,Y) );
Annotation ( A, Direction = (DX,DY) );

Artefact (B = Observation );
Annotation (B, Precision = (r, t) ) ;
Annotation (B, Error = epsilon) );

Arrow (C = Output(A,B));
Annotation(C, Bandwidth = a Mb_per_s);
```

The pseudo-code snippet in this example is (just) a suggestion in an otherwise unspecified language. Of course, the syntax of a workflow can take many other forms.

Another reason to represent workflows textually instead of graphically is related to the further goals of the management system with the workflows: valuation, selection, instantiation and monitoring. Each workflow constitutes some solution to some need. In general, multiple (candidate) solutions will be computed for a need. Furthermore, in general there will be multiple needs present in the system.
Hence, one of the tasks of the workflow management process is to select and compose a full system configuration. This implies that the management process has to

- valuate candidate solutions. The value of a candidate solution is determined by the priorities of the business process.
- select a solution from the set of candidate solutions for each need.
- compile a workflow containing the selected solutions, (at least) one for each need. Such a workflow will contain a number of parallel workflows, one for each need.
- the compilation process has to filter out all ‘impossible’ combinations of solutions. An impossible combination of solutions would be, for example, the combination of two workflows X and Y, both employing the same welding-robot, that, when combining X and Y, has to weld on two different products at the same time.

2.3 Workflow Generation & Management

The principle of automated Workflow Generation & Management is that a dedicated set of services compute and execute workflows, that is, translate the workflows to instructions for operational services. This principally divides a system into two domains, the management domain and the managed domain, as depicted in Figure 3.

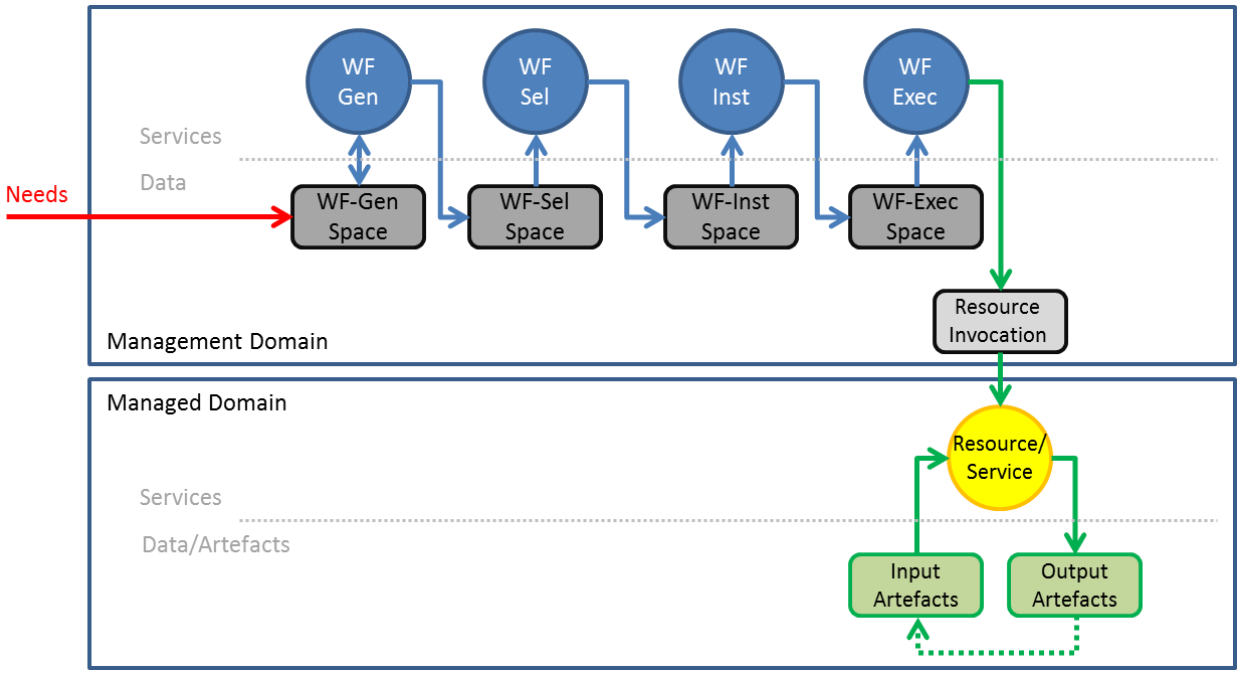


Figure 3: Workflow Generation & Management principle

The services in the management domain can be divided into generation, selection, instantiation and execution services.

- Generation services are responsible for creating workflows that produce outputs according to requirements stated in needs. Workflow generation is discussed in section 2.4.
- Selection services select those workflows that are most suitable for execution. Workflow selection is discussed in section 2.5.
- Instantiation services determine allocation of resources (or services) in the managed domain to tasks in workflows. They formulate service level agreements and operational conditions as needed. Workflow instantiation is discussed in section 2.6.
- Execution services extract information per managed resource from a workflow and translate it into invocation data that can be used to configure a resource. Workflow execution is discussed in section 2.7.

Depending on the design, some of these management services may be allocated to the same service provider(s), for example selection and instantiation may be collocated in one management resource.

Regarding the managed domain, the operational resources are organized and co-ordinated (orchestrated) according to the specifications of the workflow.

Remark: To ensure the proper behaviour of all executed workflows, which indeed includes the workflow of workflow generation & management, quality of service management is applied. Quality of service management is discussed in section 2.8.

2.4 Workflow Generation

Workflow generation is the process of iteratively composing a workflow automatically. A workflow generator manipulates partial workflows, adding workflow components until the workflow is complete. A complete workflow has a clear specification of tasks for each operation required to achieve the desired output, as well as a clear specification of the in-

and outputs of each operation. The start point (source) of the workflow is (a set of) initiating tasks or inputs, and the end point (sink) is the desired output.

Collaborative workflow generation & management (CWFGM) differs from 'plain' workflow generation and management in the sense that the generation process is distributed over a number of generators and managers. These generators and manager, in general, belong to different stakeholders, trying to achieve a common goal, the desired output, collaboratively.

Our particular interest in the context of Productive4.0 is collaborative workflow generation & management.

2.4.1 Workflow Generation Approaches

Principle

The general principle of a WF generator is to add workflow components (actions, artefacts, arrows and annotations) to a workflow-under-construction. A workflow-under-construction is called a *partial workflow*. A WF generator uses explicit or implicit domain knowledge regarding resource capabilities to specify activities in WFs.

Figure 4 presents a typical WF Generator. Stakeholders place needs, formulated as partial WFs, in a local repository of partial WFs (the WF-Gen space in Figure 3). Partial WFs contain a (possibly empty) set of desired outputs that have no activity creating these outputs (yet). The generator mechanism extracts a partial WF *pwf1* from the partial WF repository, and, using domain knowledge, checks whether it can add a component creating a desired output. If so, a new partial WF *pwf2*, containing *pwf1* and the added components, is constructed. The set of desired outputs for *pwf2* consists of the union of the inputs required by the new components and the set of desired outputs of *pwf1*, minus the desired outputs in *pwf1* that are created by the new components. *pwf1* is discarded. If the set of desired outputs of *pwf2* is empty, it is delivered as a complete WF (the WF-Sel space in Figure 3); if not *pwf2* is added to the repository of partial WFs. Trivial extensions to this typical WF generator include a test to see whether a partial WF needs to be discarded for other reasons, for example becoming too expensive; usually this indicates that infinite repetitive patterns are being generated in a WF.

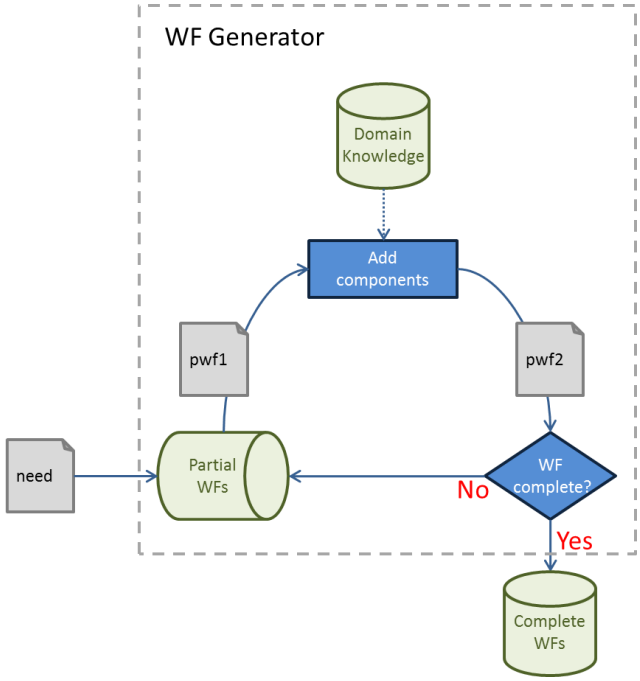


Figure 4: Typical WF Generator

Generation using propagation

An iteratively backward or forward propagating WF generator uses a conventional reasoning mechanism to add components to a partial WF. As an example consider the case where a partial WF produces a desired output *Artefact0*, by means of *Action1* and *Action2*:

Artefact2 is in the set of desired outputs for this partial WF (desired output for new components).
The WF generator finds *Artefact2* is produced by *operation X*, which requires *Artefact3* as input.
Operation X is added as *Action3*. The required input for *Action3*, *Artefact3*, becomes the new desired output.

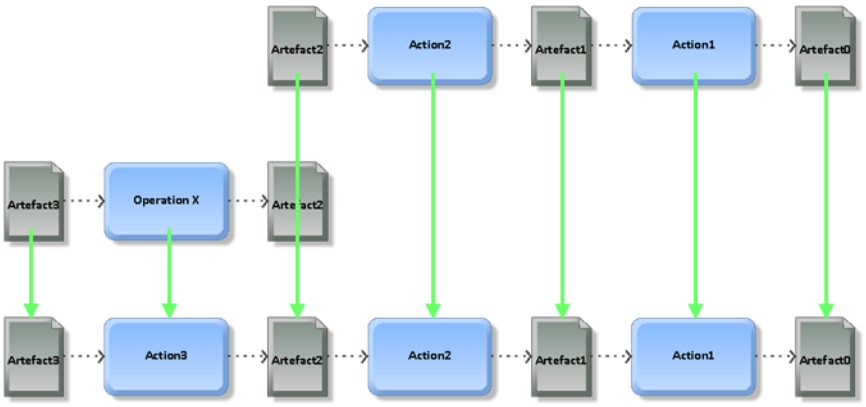


Figure 5: WF generation using propagation

Action3 describes the usage of *operation X*. Figure 5 illustrates this example; green arrows denote how elements from one WF fragment are copied into the newly generated WF fragment.
This example uses backward propagation, as the workflow is constructed 'back-to-front' or upstream: starting with the desired output (sink) it propagates back to the initiating inputs (sources).A forward propagating generator reverses this order: starting with the available inputs it adds actions producing all possible outputs.

Generation using templates

Another approach to WF generation makes use of templates: predefined WF fragments that contain gaps where other WF fragments need to be connected. The actions that need to be inserted in the gaps can, for example be, a paintjob, insertion of an optional component or a transport operation. As an example, consider the partial WF[5] that requires *Artefact0* to be produced:
The generator finds a template that produces *Artefact0*, using *operations X* and *Z*. The template is used to generate the next partial workflow. A gap requires an unknown action/sub-WF 'between intermediate *Artefact2* and *Artefact1*'. The desired outputs for components in the partial WF includes *Artefact3* and *Artefact1* (based on *Artefact2*).
The next iteration the generator finds an *operation Y* producing *Artefact1* based on *Artefact2*, and inserts it in the gap as *Action2*. The new partial WF contains *Artefact3* as desired output.

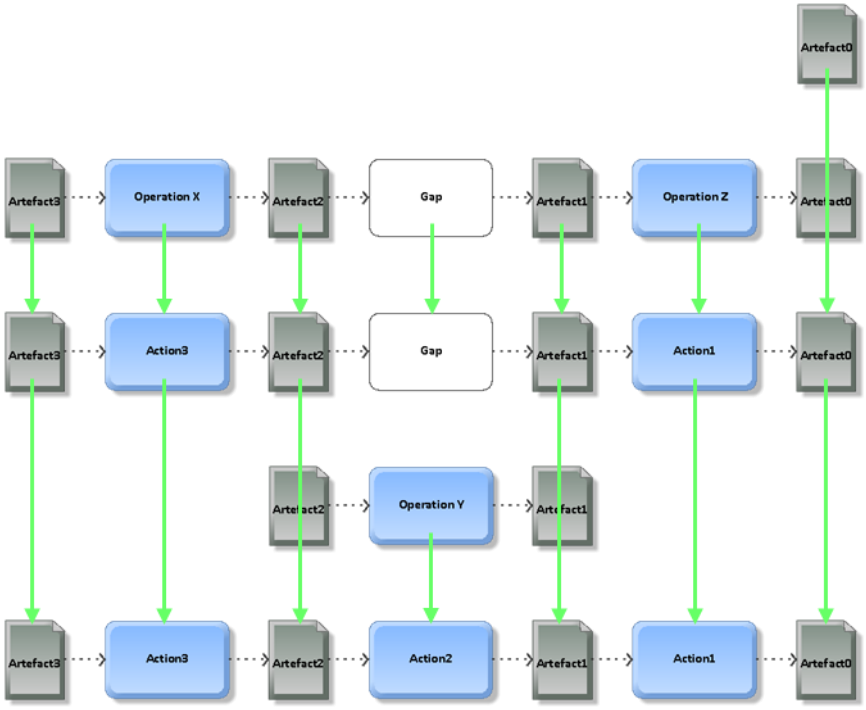


Figure 6: WF generation using template filling

[5] This partial workflow constitutes a need.

Figure 6 illustrates this example. The green arrows in this figure denote how elements of a workflow fragment are copied into a newly generated fragment.

Compositionality

Compositionality is an encapsulation concept, where some action in a WF is specified as a separate sub-WF. This action is called a 'compound action'. WF II is effectively encapsulated by the compound action, since it is represented in WF I by the compound action. In turn, WF II may contain compound tasks.
An example of compositionality is depicted in Figure 7, where a task 'B' is specified in a separate sub-workflow indicated by the red box; note that the in- and outputs of task 'B' are identical to the in and outputs of the sub-workflow are identical.

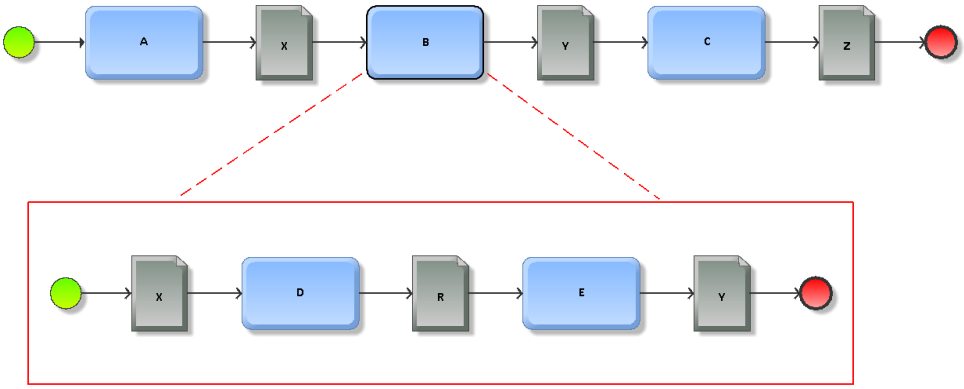


Figure 7: Example of compositionality in a workflow

Generation using templates depend on compositionality, as the gaps in the templates can be thought of as (compound) actions that still need to be specified.

Benefits and drawbacks

The benefits of propagation generators are, compared to template filling, a greater flexibility yielding a larger amount of workflows that can be generated. Backward propagating generators are more appropriate in cases where the end-goal is clearly specified, for example a set of requirements from a customer, forward propagating generators are more appropriate in case the starting position is well known and a general direction for activities can be defined, for example, traveling from your home location to a foreign destination. In addition, forward propagation can be used in case future actions cannot be determined fully before the current actions are completed.
In general, propagating generators will use more iterations than template filling generators to achieve the same workflow.

The benefits of template filling include that it is possible to generate optimal WFs for recurring needs, a natural separation of concerns, modularity and reuse.
The drawback of templates include the introduction of rigidity, which for example precludes an alternative processing for the left- or right hand side of a gap. Also, it may be more difficult to find an exact fit for the gaps in ad hoc collaborations.
Template filling generators are most appropriate in cases where operations can functionally be grouped or predominantly occur in a common order. The gaps occur where the common order is interleaved with variable operations.

From a conceptual point of view, template-filling generation can be degenerated to case where the gaps only occur at the end (sink-side) of the workflow templates. This resembles the forward propagating generation. Likewise it can degenerate to the case where gaps only

occur at the start (source-side) of the workflow templates, in which case it closely resembles the backward propagating generation.

2.4.2 Collaborative Workflow Generation and Management

Central to collaborative workflow generation & management (CWFGM) is the exchange of information for generation and orchestration purposes. The WF generation processes may use any applicable mechanism during the generation phase, the collaboration aspect is given shape during the exchange of the partial workflows. The partial workflows are shared between the WF generators, allowing each generator to add the components to further complete the partial workflow. As shown in Figure 8, this can easily be achieved by placing the repository of partial WFs outside the WF generator. Similar to the single WF generator case in Figure 4, once the set of desired outputs of *pwf2* (or *pwf2'*, respectively) is empty, it is delivered as a complete WF (the WF-Sel space in Figure 3).

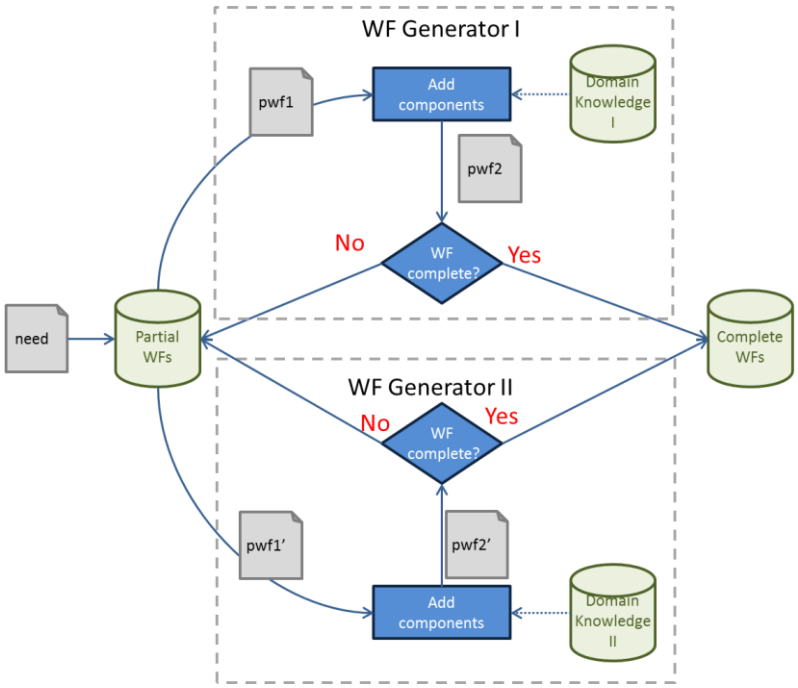


Figure 8: Typical collaborative WF generation

At the same time, we can derive a number of principal requirements for CWFGM:

1. The generators need to have access to a data sharing facility, the partial WF repository.
2. The generators need to adhere to the same format, representing partial workflows in the partial WF repository.

An abundance of data sharing mechanisms is available for storing partial WFs, for example Vortex OpenSplice [73], Hazelcast [74] or Redis [75].We will consider this aspect out of scope for this report.

Format

Collaborative workflow generation relies on agreements of protocol, syntax, grammar and semantics of representation and exchange between the generation and management processes.

Syntax, grammar & semantics

The syntax and grammar of the representation determines how (partial) workflows are formulated; syntax determines the spelling of tokens in the representation, while the grammar specifies how the elements in the representations are ordered.
The semantics of the representation determines what a representation means, that is, how the representation is to be interpreted.
The combination of syntax, grammar and semantics constitutes a language. Chapter 4 investigates WF languages in detail.

Protocol

The protocol determines how generators have access to the partial workflows, how various iterations of partial WFs are distinguished and what happens with complete workflows (storage, further processing). These are design choices, and will be discussed in chapters 4.3 and 0.

2.5 Workflow Selection

As discussed in section 2.2.2, we can associate a value to each workflow, given some policy, compare the values of individual workflows, generated in response to the same set of requirements, and select the most desirable workflow available for a need. In general the workflow selection service will select a feasible candidate workflow from an ordered set of candidate workflows, based on some policy.

Feasibility

Feasibility revolves around the issue whether a workflow, although technically executable, is also actionable for real. This means, for example, to check whether there is at least one known service provider, providing service with sufficient quality, for each task in the workflow. Secondly, some policies might prohibit the execution of a workflow even though service providers are available for each task; as an example consider the case where two competing agencies do not want to be involved in the same workflow.

Valuation

The purpose of valuation is that candidate workflows can be ordered according to some criterion. This criterion is usually grounded in a relevant quality of the produced output. Valuations of workflows can be based on a number of workflow characteristics, for example:

- amount of tasks in the workflow
- computation based on costs of all tasks in the workflow, provided a cost for each task is given. The cost for a task can be expressed in, for example, money, time or consumed bandwidth. Depending on the type of cost, the computation is different, for example in case of money, the total is an addition of the cost of all tasks, whereas in case the cost is time (throughput) the cost of a workflow is the amount of time required for the critical path in the workflow.
- quality of the produced output, for example the amount of error or uncertainty in the end-result.

Policy

In the context of workflow selection, a policy constitutes a predefined and previously agreed upon set of rules to determine the best workflow from a set of candidate workflows. Commonly policies aim to select a candidate based on feasibility, cost or another relevant quality of the produced output. Policies can state to select, for example:

- the cheapest candidate given a cost criterion,

- the best candidate, given a quality criterion,
- a combination of cost and quality, for example the best candidate below maximum cost, or the cheapest candidate with a minimum quality,
- any other rule using cost and qualities or other relevant workflow aspects that unambiguously leads to selection of a candidate.

2.6 Workflow Instantiation

Workflows are generated for the purpose of instantiation: as an actionable business process, executing the workflow will yield the output that corresponds to the goal of the business process. In this section we will investigate how information included in the workflow can be used to instantiate and orchestrate the production of the desired outputs.
In section 2.2.2 we identified five elements that document relevant information in workflows: actions, artefacts, arrows, actors and annotations. Let us consider a simple workflow as in Figure 9.

Figure 9: A simple workflow composed of actions and artefacts

In addition to the basic elements actions, artefacts, arrows and resources, a workflow may contain annotations to these basic elements. During the generation phase, information regarding requirements, criteria and conditions regarding the operations (actions), in- and outputs (artefacts) or exchanges (arrows) can be available; they are a part of the capability descriptions generated during the design of the capability. As this is useful information for the other CWFGM and QoSM processes, they are included in workflow in the form of annotations.
In our example, in Figure 10, the artefacts ‘a’ and ‘b’ of a simple workflow have been annotated with quality requirements, for simplicity represented by ‘QReq(A)’ and ‘QRec(B)’. This means that at the time of generation, the generator found an operation that produces a desired output, artefact ‘b’, satisfying a number of requirements, documented in ‘QReq(B)’. These requirements can state cost, time, precision, or whatever is relevant in the context of the workflow.
To have ‘b’ produced to requirements in ‘QReq(B)’, service level conditions for action ‘B’ have been documented in an annotation, denoted in Figure 10 by ‘SLc(B)’. These service level conditions may document some configuration information, or conditions regarding the quality of the resource to be allocated. At the same time, quality requirements have been propagated or transposed to the input of ‘B’, artefact ‘a’, in the form of annotation ‘QRec(A)’.

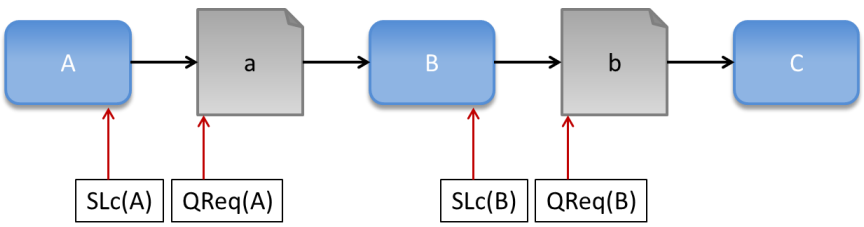


Figure 10: Quality annotations in a simple workflow

In the next iteration of WF generation, the operation satisfying the requirements in 'QReq(A)' has been found and documented in action 'A'. In the same fashion as with action 'B', the service level conditions for action 'A' have been documented in annotation 'SLc(A)'.

After generation an appropriate workflow is selected for instantiation. The selection is based on the criteria of 'most suitable', where 'most suitable' depends on current priorities, agreed-upon policies and other activities deployed by the stakeholders. The current priorities originate from the customer requirements and business goals of the stakeholders. The agreed-upon policies are constituted by the consortium of stakeholders, responsible for producing the desired output or customer service. Policies define the rules of engagement between stakeholders and between stakeholders and customer. The other activities deployed by stakeholders may render a workflow unfeasible, for example because critical resources cannot be made available in the timeframe required by the workflow.

For the selected workflow available resources, actors, are allocated to actions in a workflow; Figure 11 depicts example allocations, denoted by yellow circles in our example workflow.

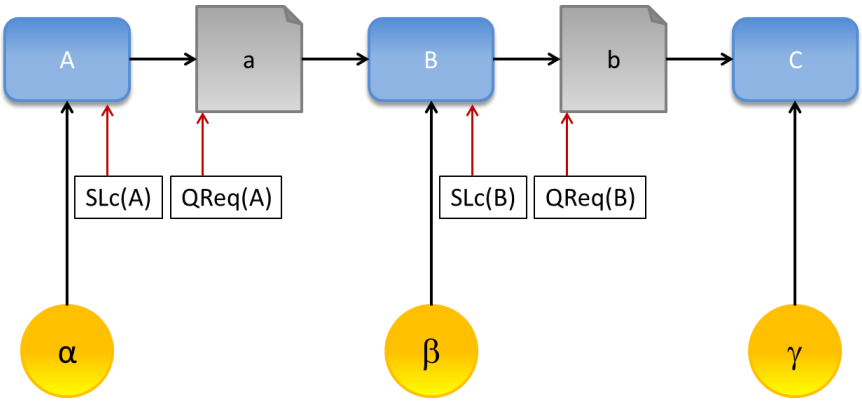


Figure 11: Resources allocated to actions in a simple workflow

In this figure, action 'A' is to be executed by resource 'α'. In the presence of the quality requirements on the execution of action, actors will be selected based on their ability to satisfy these constraints. Along with the addition of the resource-allocation to the workflow description, a service level agreement (SLA) will be generated and stored in an annotation. The contents of the SLA will be based on the QoS requirements documented in the 'SLc()' and 'QReq()' annotations mentioned earlier.

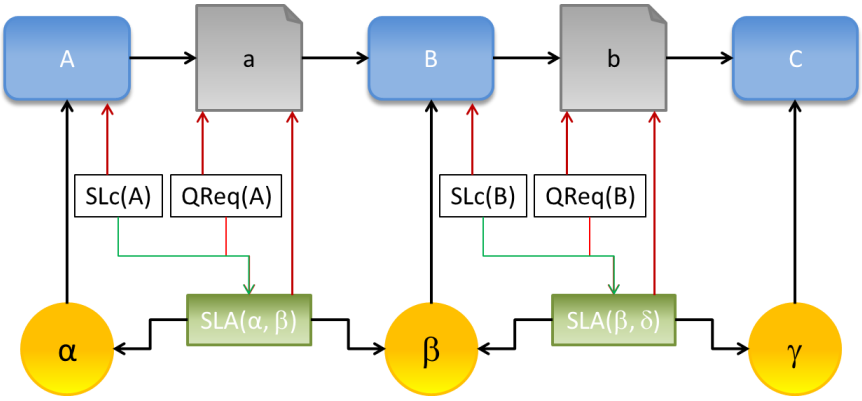


Figure 12: Service Level Agreements between allocated resources in a simple workflow

Figure 12 depicts the SLAs between 'α' and 'β' and between 'β' and 'γ' as green boxes 'SLA(α, β)' and 'SLA(β, γ)' respectively. Since the SLA concerns the production of, for example 'b', the annotation will be associated to artefact 'b' in the workflow. In addition,

the SLA will explicitly refer to the allocated resources 'β' (as the service provider) and 'γ' (as the service client).

2.7 Workflow Execution

Workflow execution takes an actionable workflow to be executed (from the WF-Exec space in Figure 3) and translates the elements related to each allocated resource into resource invocation data.

Depending on the resource to be invoked, the workflow execution service either transmits the data to the resource, or if the resource needs to be started, starts the resource with the appropriate configuration.
Each workflow execution service provider manages a group of operational resources, ranging from all operational resources (fully centralized model, see Figure 13) to one resource (fully distributed model, see Figure 14).

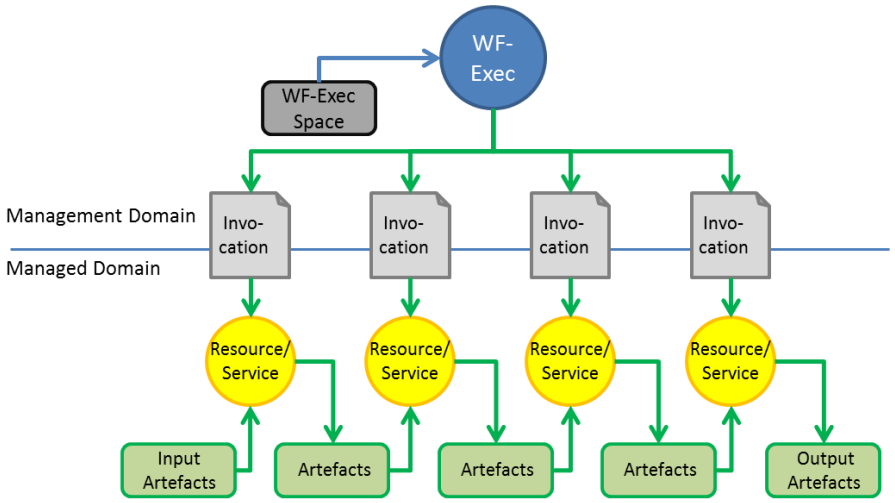


Figure 13: WF execution: fully centralized model

In the fully centralized model one WF execution services generates resource invocation data for all resources. This typically implies that there is a limited variation in the resources, and that all resources belong to the same agency. The other end of the spectrum is the fully distributed model, where a workflow execution service provider is associated (or even included in) each operational resource.

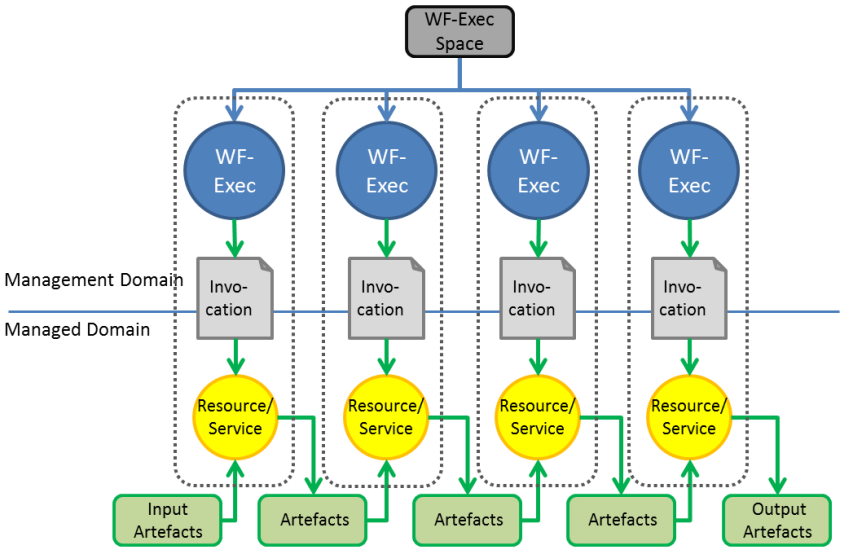


Figure 14: WF execution: fully distributed model

The fully distributed model is more applicable when there is a large variation in the operational resources or multiple agencies are involved in the executed workflow. The same executed workflows can be realized with either the fully centralized or fully distributed model. Additionally, mixed models are possible, where each WF execution service provider manages a (limited) number of resources.

2.8 QoS Management

The purpose of quality of service management in the context of ePLM is to ensure that the results described in a workflow are actually achieved with the quality required, or if that is not possible for some reason, identify the problem and apply a mitigation. This section discusses the vital components that enable QoS management. These vital components are:

- Deriving the proper information from a workflow with allocated actors and service level agreements to make the execution manageable.
- Monitoring of the instantiated workflow, that is sample the execution of operations and results for quality and performance.
- Identifying issues that prohibit achievement of the desired results of the workflow.
- Mitigate the issues found.

Extending the workflow generation & management principle from section 2.3, QoS management samples workflow execution and mitigates exceptions according to the principle depicted in Figure 15.

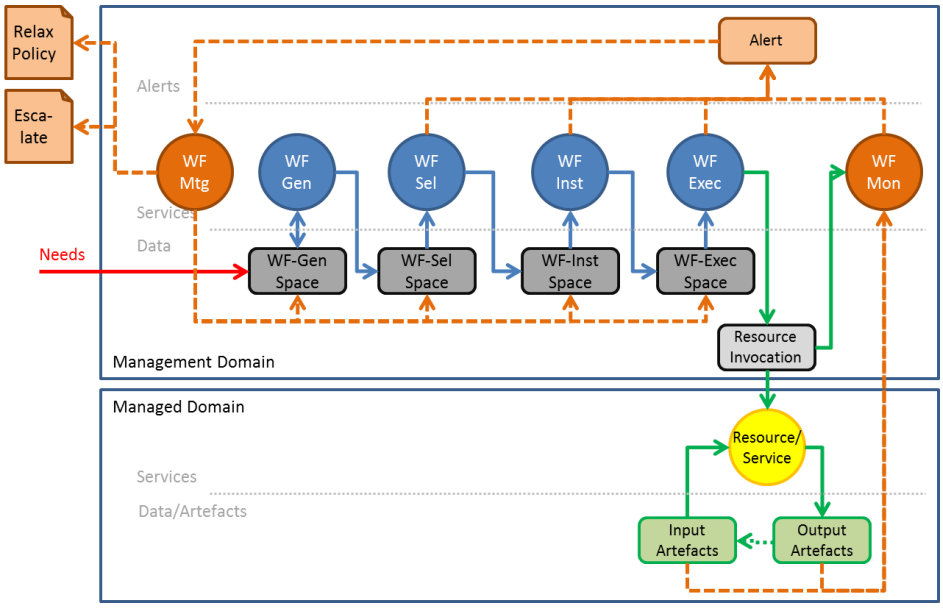


Figure 15: QoS management principle for an orchestrated system

As depicted in Figure 15, anomalies in the operational workflow(s) executed, as well as the WF generation & management workflow are used to raise alerts.

Note that the monitoring service processes, WF Mon, are configured and instantiated like any other service resource in the system. The alerts are processed by a mitigation service, WF Mtg, which, depending on the situation, inserts an intervention description in any of the WF Management data spaces, or sends an elevated alert to the stakeholders; the elevated alert can include a request for

- a relaxation of (some) policies, since the current policies are prohibiting production of the desired output
- escalation, since the current set of operational resources is insufficient or inadequate to produce the desired output.

Section 2.8.1 discusses how the information required to make instantiations manageable is derived and formulated for the purposes of monitoring and mitigation; it also discusses how monitoring can be achieved and how triggers for the mitigation process can be generated. These triggers need to provide sufficient information for the mitigation process to execute the appropriate response. Finally, section 2.8.2 discusses mitigation options and strategies that are applicable to the domain of ePLM.

2.8.1 Monitoring

The workflow generation mechanisms discussed in section 2.3 and the workflow instantiation process discussed in section 2.6 yield an actionable workflow. This actionable workflow includes information regarding the service levels required and provided by the actors. The next step is to derive the information that enables the QoSM processes to monitor the execution of a workflow and mitigate in case of deviations.

The information required for monitoring is included in the SLA's in the workflow. It is not uncommon for an SLA to include the requirement as well as the 'sanction'. This means that the monitor can be configured using a recipe, containing condition-action pairs. Each condition-action pair includes the event or circumstance (the condition) that triggers the sanction (the action) to be executed.

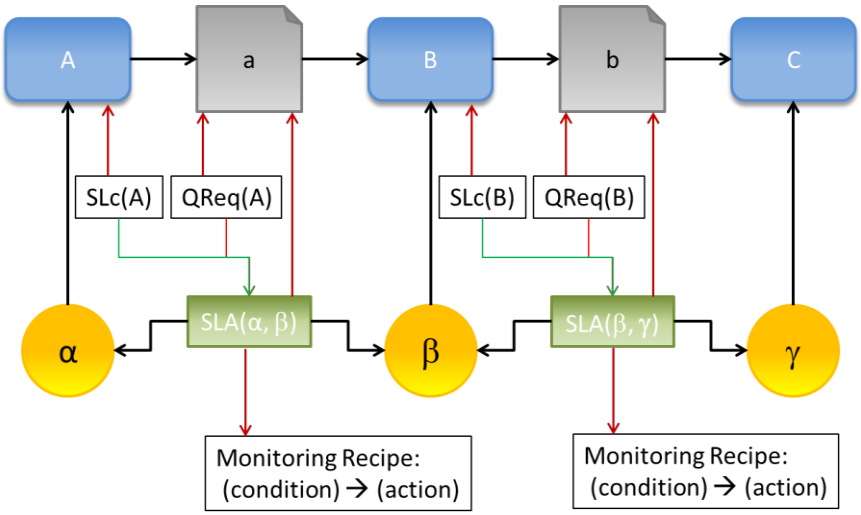


Figure 16: SLAs in a workflow provide the required information for monitoring recipes

Finally, each monitoring recipe is assigned to a monitoring agent. A monitoring agent can be implemented using a skeleton process that is configured using a monitoring recipe.

The workflow is instantiated by issuing the appropriate configuration messages to the resources and services allocated to activities in the workflow (initiation). Furthermore, monitoring agents are initialized, which monitor workflow execution aspects as specified by the generated monitoring recipes. Technically, generating the monitoring recipes and configuring the monitoring agents is a responsibility of the CWFGM processes.

Monitoring Agents sample progress and service levels according to their recipe. A monitoring recipe will have the general form of a list of "{<condition>, <action>}" pairs. The condition represents a desirable or undesirable event or situation for which action is required. If the condition of a condition-action pair holds, the monitor will execute the corresponding action. A typical action of a monitor would be to send a message, containing sufficient information for the QoS management service to mitigate an undesirable situation (Figure 17).

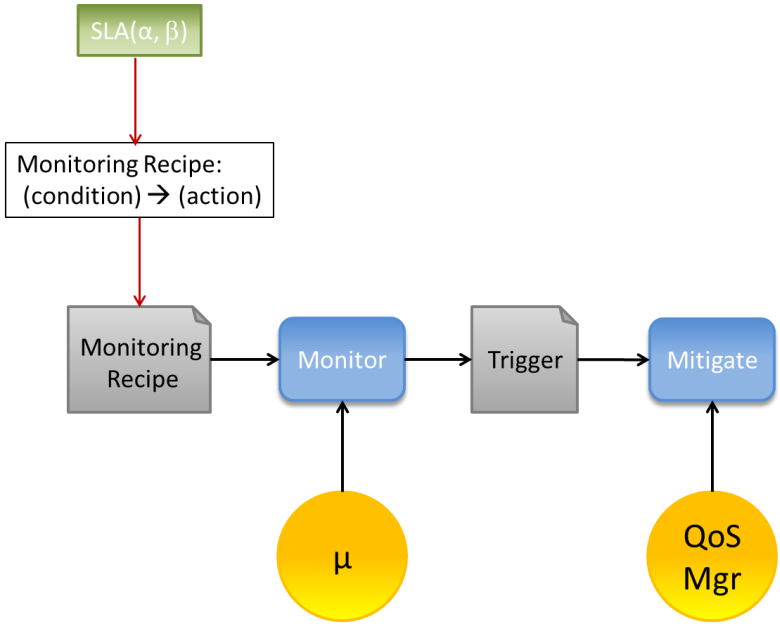


Figure 17: QoS Management: the monitoring and mitigation process

A second use of the monitoring agent is the function of triggering agent for a new phase in phased workflows. As soon as a milestone in the workflow execution is achieved, the monitoring agent can issue a trigger that causes the generation of a workflow for the next phase of a process. Monitoring agents can also be used for the bootstrapping phase of the CWFGM services, following pre-generated recipes to monitor whether the need for WF generation emerges, based on the occurrence of a new need.

2.8.2 Mitigation

The mitigation service is triggered by the alerts inserted in the Alert space in Figure 15. The aim of the alerts is to explain to the mitigation service that the CWFGM service failed and what caused the failure. An alert therefore should contain all the information necessary to determine a mitigation response. In principle, an alert contains the aspects *Why*, *Who* and *What* of the failure that was detected:

- *Why*: Why did the workflow processing fail? This aspect identifies the category of failure, for example, lack of resources.
- *Who*: Who (what entities) caused the process to fail? This aspect identifies the elements in workflow that cannot be resolved, for example, a task A cannot be allocated to a resource.
- *What*: What are the details of the failure? This aspect identifies the details of the failure, for example, task A requires a service quality level for some quality Q that cannot be achieved by any resource in the system.

It is obvious there are different types of alert and these will lead to different types of response.

Types of alert

The WF Selection, WF Instantiation and WF Execution services can detect that the results produced in the previous step cannot be further propagated and raise an alert indicating this problem:

1. The WF Selection service may find that all the workflows produced require capabilities that are currently not available in the system. The alert (type: **Incapable**) will identify the missing capabilities for each workflow generated and the mitigation service will decide to generate an "Escalate" request for the stakeholders (see Figure 15), requesting to solve this issue by escalation, that is, attracting new resources with the required capabilities.
   The response to this trigger has to be authorized and generated outside the scope of the CWFGM services.
2. The WF Selection service may find that the integrity constraints defined prohibit execution of all the workflows produced. The restricting policies will be identified and included in the alert (type: **Too Strict**) for each workflow produced. The mitigation service will raise a “Relax Policy” request, requesting the stakeholders to relax some of the identified policies (see Figure 15), thus allowing execution of one of the workflows. The response to this trigger has to be authorized and generated outside the scope of CWFGM services.
3. The WF Instantiation service may find that (some of) the resources possessing the capabilities required in the workflow, are unavailable, for example, due to being allocated to other tasks. This prevents the allocation of such resources in case the timing constraints in the workflow do not match the current schedule of the available resources[6].

[6] On a side note, consider the case where the agencies have specified a strategy that prioritizes some goals of the collaborative effort over others. In that case, if the current workflow has a higher priority

The WF Instantiation service will generate an alert (type: **Allocation**) identifying the resources that cannot be made available (in time) for contribution to the workflow execution. The mitigation service will trigger the WF Selection service to select another workflow if possible, and if not, raise an escalation trigger requesting resources with the required capabilities.

4. The WF Execution service may find that it cannot execute the instantiated workflow, because the resources or monitoring services respond inappropriately to the issued Invocation recipes[7]. The Execution service will generate an alert (type: **No Response, No Monitor, Rejected** or **Mon Rejected**, depending on the situation), identifying the 'misbehaving' resources or monitoring agents.
   The mitigation service will try to work around this situation by issuing a re-instantiation request, identifying the resources to be avoided in instantiation. The CWFGM process then reiterates from WF instantiation.
5. During normal operation, if the Monitoring agent finds a deviation in performance or provided service quality, it will generate an alert (type: **Progress, Performance** or **Evaporation** (evaporation is the case where WF generation service cannot find candidate workflows, due to lack of capabilities: the required capabilities do not exist in the current consortium. The detection of this failure involves tracking of the partial workflows generated for a particular need. The mitigation for this failure is escalation, that is the extension of the consortium with agencies that do possess the missing capabilities.), depending on the situation) specifying the deviations detected. The mitigation service will respond to this alert by requesting a new instantiation for (part of) the affected workflow, indicating the resources to be avoided; the CWFGM process then reiterates from WF instantiation.

There is no alert defined for failure during WF generation. This is a deliberate decision: WF generation mechanisms will frequently find that they cannot extend a partial workflow. This also does not necessarily mean that the workflow generation process failed, but rather a consequence from the separation of concerns principle: other workflow mechanisms involved, accessing a different set of knowledge, may find that they can extend the partial workflow. Furthermore, none of the WF generation mechanisms has an overview of the entire WF generation process, or the transitive closure of all the knowledge currently contained. However, this does not mean there are no failures to be detected. Two possible failures that prohibit successful the WF generation are evaporation and cyclic dependency.

Cyclic dependency is the case where WF generation reaches a stalemate situation, in brief: a task A requires task B, which requires task A. The basic principle of such a circular dependency may take very complicated forms that are very costly to identify. The result of this situation is that the WF generation mechanisms will keep expanding the partial workflows indefinitely, without ever producing a complete workflow. A mitigation for this type of failure is to limit the maximum allowable size of partial workflows in terms of the amount of tasks they may contain. Surpassing this limit will automatically lead to discarding the partial workflow. The limit can be set using a policy.

---

than the conflicting workflow (already being executed by the required resources), the decision might be to re-allocate the resources to the current workflow, and abandon the execution of the lower priority workflow; this leads to another situation, where mitigation by the Adjustment service is required as a result of the inability to execute a workflow (in this case the lower priority workflow) due to unavailability of resources.

[7] This situation indicates that the resource or service requires intervention from an agency's operators to resolve, but here we focus on the mitigation by the mitigation service.

Types of response

The alerts generated by CWFGM and monitoring services trigger a response from the mitigation service, a mitigation; a mitigation is a (request for) modification of:

- the generated, selected or instantiated workflows. The mitigation action requires new workflows to be generated, another workflow to be selected or the selected workflow to be instantiated differently.
- the policies and integrity constraints governing workflow generation, instantiation and execution. The problem is that in this case the policies or integrity constraints are too restrictive to solve the needs in the system
- the composition of the system. The problem is in this case that the system lacks the knowledge and capabilities to solve the challenges in the operational scenario.

Table 1 associates types of alert with types of response. This table identifies for each case, the originator for the alert, the key information elements in the alert and the type of response from the mitigation service.
The key information elements in the alert contain three components:

- a component **WF-element**, pinpointing the source of the problem in the workflow. The value for this component can be an *action*, an *artefact* or an *actor*.
- a component **required**, identifying what had to be associated to the WF-element. Possible values of this component are description of an *action*, an *artefact*, an *actor*, a *quality* or a *value*.
- a component **obtained**, describing what was actually produced. Possible values for this component are descriptions *'none'*, of a *quality*, of a *value* or of an (policy) *instance*.

#	Alert type	Source	Key Information elements			Response type
			WF-element	required	obtained	
1	Incapable	Selection	action	resource	none	Escalation
2	Too strict	Selection	action	quality	instance	Relax Policies
					quality	Relax Policies
3	Allocation	Instantiation	action	resource	instance	Re-do Selection
					status	Apply prioritization
					schedule	Escalation
4	No response	Execution	resource	value	none	Re-do Instantiation
5	Rejected	Execution	resource	value	value	Re-do Instantiation
6	No monitor	Execution	resource	value	none	Re-do Instantiation
7	Mon rejected	Execution	resource	value	value	Re-do Instantiation
8	Performance	Monitor	action	quality	instance	Re-do Instantiation
					quality	Re-do Selection
						Re-do Generation
9	Production	Monitor	artefact	quality	quality	Re-do Instantiation
						Re-do Selection
						Re-do Generation
10	Evaporation	Monitor	artefact	artefact	none	Escalation

Table 1: Failure Trigger Mitigation

Explanation of alerts and responses

This section briefly explains the contents of Table 1. The explanations are itemized according to the index in Table 1.

1. **Incapable**: The Selection service encountered in all of the candidate workflows a task that could not be performed by any of the capabilities of the registered resources. The key information elements in the alert are filled out as:
   - WF-element contains the action element that specifies the unresolved capability
   - required contains the resource capability description that was searched for.
   - obtained contains the value "none", since no adequate resource was found.

   It is evident that the system lacks resources and must attract new agencies (and/or resources) that do provide the required capabilities, hence escalation is the appropriate response.

2. **Too Strict**: The Workflow Selection service encountered in all of the candidate workflows a task that could not be performed by any of the registered resources, given the current requirements or policies. However, there are resources that possess the capability specified by the task. The key information elements contain:
   - WF-element: the action element that cannot be allocated given current conditions.
   - required: the annotation containing the quality description that cannot be satisfied.
   - obtained: In case a policy is hindering allocation, the policy instance that is too strict. In case a resource can perform the required capability in the WF-element, but not with the required quality (required), the quality constraint of the candidate resource that could be allocated with relaxed requirements.

   In this case the policies/requirements that are currently in place are too restrictive to obtain a solution. By relaxing the policies or requirements, a candidate workflow can be selected. Hence a request for relaxation is the appropriate response; if policies or requirements cannot be relaxed, the system needs to be augmented with appropriate resources (escalation).

3. **Allocation**: A candidate workflow has been selected, since all capabilities required in the workflow are available in the system and it was the best available candidate workflow given the current policies and conditions. However, for some action, the Instantiation service cannot find a resource that is available at the time or under conditions specified in the workflow. The key information elements contain:
   - WF-element: the action element in the workflow where a resource cannot be allocated.
   - required: the specification of the resource requirement, complete with the conditions to be satisfied for allocation.
   - obtained: a (list of) resource instances with the resource availability status, given the current planning in the system (leading to mitigation response i, ii or iv), or alternatively, the (integrity) constraints of the resource that prohibit allocation (leading to mitigation response iii or iv).

   There is a number of mitigation responses possible for this case:
   i) The easiest mitigation is to re-do the workflow Selection process, excluding the selected workflow as a candidate (Re-do Selection).
   ii) If policies are in place that regulate the priority of activities and goals in the system, a resource currently allocated to a lower priority workflow is re-allocated to the current workflow. Of course, the lower priority workflow will now cause another alert to be generated (Apply Prioritization).
   iii) Policy relaxation, if resources are available, but are prohibited from allocation due to their integrity constraints (Relax Policies).
   iv) Escalation, to request new resources that do provide the capability, while satisfying all stated constraints (Escalate).

   The actual mitigation is selected by the WF Mitigation service, based on the current mitigation policy of the system.

4. **No Response**: The WF Execution service does not obtain an acknowledgement (or does not obtain an appropriate acknowledgement) from the resource targeted during invocation. The cases of resource invocation failure and monitoring invocation failure are treated identical; the type of resource causing the problem is a common resource in the resource invocation failure case, and a Monitoring agent in the monitoring invocation failure case. This usually indicates that the resource or monitoring agent is no longer available, due to connection problems or catastrophic error in its mechanism. The key information elements contain:
   - WF-element: the action in the workflow and the allocated resource (identifier).
   - required: the expected acknowledgement message.
   - obtained: "none", or alternatively, the response received.

   This situation (irresponsive resource) requires a separate mitigation (which can be kicked off as a result of this alert. The mitigation to this problem is to re-do the WF Instantiation process, excluding the failing components from allocation Re-do Instantiation).

5. **Rejected**: The WF Execution service received a rejection message from the resource targeted during invocation. The cases of rejection during resource invocation and monitoring invocation are treated identical.
   The problem detected is that the resource (monitoring agent) to be invoked responded to the invocation with a rejection message, indicating it would not execute the requested task. The key information elements contain:
   - WF-element: the action in the workflow and the allocated resource (identifier).
   - required: the expected acknowledgement message.
   - obtained: the response received, rejecting the assignment.

   This means that the resource (monitoring agent) rejected the assignment, probably because it is busy doing an unknown activity. It also means that the resource (monitoring agent) did not update its resource status information, which might be a another point of attention.
   The mitigation response to this alert is to re-do the WF Instantiation process, excluding the busy components from allocation.
6. **No Monitor**: see item 4.

7. **Mon Rejected**: see item 5.

8. **Performance**: The monitoring agents have detected that the execution of an action in a workflow does not meet the requirements in the Service Level Agreement. The key information elements contain:
   - WF-element: the action in the workflow and the allocated resource (identifier) that fail to meet requirements.
   - required: A quality of service description from the Service Level Agreement governing the execution of the action, that was violated.
   - obtained: A quality of service description of the achieved level of service during the execution of the action.

   Depending on the severity of the failure, a re-do of the WF Instantiation, Selection or even Generation process may be required.

9. **Production**: The monitoring agents have detected that the production of an artefact in the workflow does not meet the required quality level. Usually, the deadline for delivery of the artefact is not met. The key information elements contain:
   - WF-element: the artefact in the workflow that failed to meet requirements.
   - required: A quality description from the Service Level Agreement associated to the production of the artefact, that was violated.
   - obtained: A quality description of the achieved level of quality of the artefact.

Depending on the severity of the failure, a re-do of the WF Instantiation, Selection or even Generation process may be required.

10. **Evaporation**: The monitoring agents monitoring the WF generation process, have detected that that generation of partial workflow for a WF Generation request has terminated without producing candidate workflows that can be selected, instantiated and executed (Evaporation). The key information elements contain:
    - WF-element: The artefact that cannot be resolved, since there is no knowledge regarding what capabilities produce it.
    - required: The required input Artefact(s).
    - obtained: “none”, indicating the artefact cannot be produced with the current WF generation knowledge.

    This implies that the current knowledge accumulated in the WF Generation service is insufficient to solve the WF Generation request. Hence, new agencies, providing the proper knowledge and capabilities need to be attracted (Escalation).

3. Requirements on WFGM and QoSM in Productive4.0

3.1 Introduction

The purpose of the Productive4.0 T9.2 use case, "Extended Product Lifecycle Management Best Practice", is to develop and detail the concept of a generic Digital Product Footprint (DPF) as the digital representation of a product and all of its aspects that need to be managed over time, from its conception to its end-of-life. The scope is the chosen (and sometimes imposed) responsibility of the product-owner, i.e. the company that is responsible/claims ownership for putting the product on the market.
The use case explicitly deals with the fact that a complex product will contain multiple parts that are products of other product-owners and thus that managing a complex product is a multi-stakeholder operation.

A further purpose of the use case is to build a prototype of a DPF and use this in a number of simulations in which change-events impact one or more data-items; workflows from different stakeholders must ensure that the DPF contains at all times a complete and coherent description of all relevant aspects of the product that is managed. We will therefore also have to define what the terms complete and coherent mean and how these properties of a DPF can be verified.

The prototype DPF will contain an anonymized representation of an existing complex product, involving a multi-stakeholder supply network and covering as many lifecycle phases as possible, extended with other data-items to cover a number of business processes involved in managing the DPF. The DPF will therefore consist of many different data-items, such as design data (software, hardware and mechanical parts), manufacturing data, supplier data, supply and logistics process and network data, data regarding the operational performance, the maintenance and the support of the product. The change processes for this product will include minor and major impact events, such as a mid-life upgrade, a change of ownership, export restrictions, obsolescence management, and changes in the supply network.

To enable the simulations, a demonstrator will be built that will allow for the controlled execution of changes to a prototype DPF and the inspection of that DPF.

3.2 Requirements Elicitation

Architecture Requirements		
	Autonomy requirement	
		Decoupling of a DPF from the processes that manage it: In a data space with a large diversity of data-items the duplication of data-items needs to be prohibited. On the other hand, it cannot be avoided that a single data-item is part of more than one complex data item. And that these complex data-items may be managed by independent processes and that these processes may change over time. To cope with the dynamics on the management process side, the data space should not be an integral part of the software that executes the product management tasks. Hence a complete separation of concerns and implementation between the management logic and the data-space logic is necessary.
	Sustainability requirement	
		For this requirement, the reasoning is similar to the one about the Autonomy. In a data space with a large diversity of data-items the duplication of data-items needs to be prohibited. However, single data-items can be part of more than one complex data item. Because the future management of product-related aspects may demand the construction of new complex data-items, sustainability of the DPF dictates that is it must be possible to add an arbitrary number of complex data-items in which the same data-item is a part, without any consequence for the management of earlier complex data-items.
	Scalability requirement	
		The scalability requirements covers both the physical scalability of the data space and the functional scalability of the number and type of management processes that act on the data space and the data items and the accessibility constraints that are in effect at any moment in time. Another important scalability issue is that individual data-items can have arbitrary size
	Distribution of a DPF requirement	
		Because a product has multiple parts, it is possible that these parts have different owners and hence, that the data that describes these different parts is originally generated and stored in different places. The construction of a DPF should not necessitate that all these individual data items must be copied to a central storage place.
	Notification of Change requirement	
		From the perspective of a DPF, every data-item is part of a string of linked data items. The last added element of that string has one of two states: ‘last known’ or ‘obsolete’. All previously added data-items in the string have the state ‘changed’ and a link to the next linked data-item. This notion of state and a state change are strictly local and apply only to the individual data-item. All changes must be logged as a transaction notification of that data-item and all notifications must be accessible to whoever has access to that data-item.
	Separation of concerns requirement	
		For complex and long-lived products we can distinguish at least two use cases: a small series use case and a large series use case. In the small series use case the DPF for a single product is one data space in which the data describing generic product and all copies of that product are stored and managed. In the large series use case we use one DPF to store the data related to the generic product, but every unique copy of that product will have its own DPF, hence the DPF concept must allow the construction of an arbitrary number of bi-directional relations between the DPF of the generic product and the separate DPFs of the instances of that product. As an example, consider a radar systems manufacturer as typical for a small series use case and a car manufacturer as typical for a large series use case. The separation of DPFs

		will probably be at the point of the As Delivered list-of-materials.
DPF Processes support requirements		
	Coverage requirement	
		The DPF must be able to support and capture the data generated for all relevant business processes. These processes include but are not limited to: ○ Pilot Product policy process ○ Control Product competitiveness process ○ Manage the product lines strategy process ○ Product Lifecycle Management process ○ Manage Bid ○ Manage Project ○ Procure, Make and Deliver ○ Prepare, Make and Deliver Customer ○ Control export ○ Manage Configuration refer to an existing business process (online or in literature)

4. Workflow Representation: State of the Art

In chapter 2 we have presented the baseline principles of workflows and how to use them in the context of workflow generation and management. As documented in chapter 2, Workflow Generation & Management (WFGM) and Quality of Service Management (QoSM), involve a number of activities that exchange a representation of a (partial) workflow. Therefore, we need to select a workflow representation language that satisfies the requirements of the activities in WFGM and QoSM.

In this chapter we will list the requirements on workflow representation languages in section 4.1. We will then discuss a number of candidate languages in section 4.2.

4.1 Requirements on Workflow Languages

A workflow (WF)language specifies the contents of expressions (the representation of the WF).We envision the expression will contain (groups of) components, containing elements, containing values, as in Figure 18.

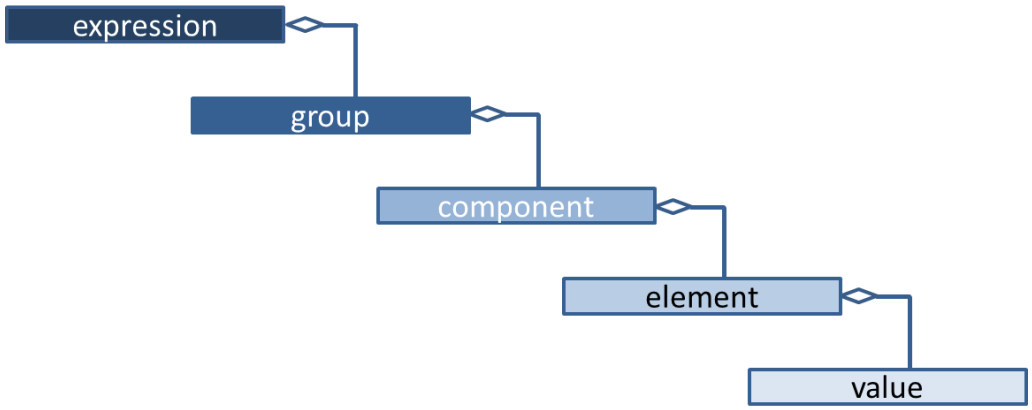


Figure 18: Generic WF language structure

For example, an expression in a WF language consists of the groups 'prologue', 'management data', 'body' and 'epilogue'. The group 'body' consists of 'action', 'artefact' and 'arrow' components. An 'action' component consists of the elements 'identifier', 'name', and 'quality'. A 'name' element could contain values such as 'multiply', 'subtract', 'paint' or 'assemble'. Each group, component and element, in general, will have its individual syntax.

It is obvious that requirements on the representation of workflows (i.e. the syntax) originate from the purpose of the representation. Besides the ability to unambiguously represent a (partial) workflow, we want the language to be actually useful in a collaborative workflow management context. This means the representation should provide sufficient handles and footholds for the collaborative generation and management processes to operate efficiently (WR.1). In extension, this implies it should be easy to read (or parse) and extend a WF representation (WR.2), it should be compact (not require large expressions to represent common elements, WR.3) and allow compositionality, i.e. allow a (reference to a) workflow be used as a workflow component (WR.4).

At the same time, it should allow the inclusion of domain specific qualities and specifications (WR.5). In the context of a dynamic and evolving trend, such as the IoT, it is worthwhile to have a language that is extensible (i.e. be able to include new terminals and constructs, WR.6).

4.1.1 Requirements elicitation

Workflow Representation Requirements		
	WR.1	**Fit for collaborative context**
		The WF language shall allow the representation to include a dedicated management component, specifying attributes and values of the represented WF relevant to the WF Management services.
		This requirement provides the generation and management processes with a location to store generation and management information. For example, the cost of a workflow in terms of operation throughput could be documented in this element. Storage of such information has two benefits: *1. efficiency: the management services don't need to calculate this value for themselves* *2. unambiguity: an approach where the management services calculate values of a workflow (themselves), different services may obtain different results (due to different implementations)* *Other entries in this element may include: identifier of a workflow, status of the WF(partial, selected, allocated, execution, obsolete),priority, problem-owner , unresolved branches in the workflow, other cost values associated to a workflow (amount of resources, euro's, precision, ...), etcetera.*
	WR.2	**Fit for Generation**
		A WF language shall support the WF generation process, by formulating components in ASCII only and listing components in an order-independent fashion.
		This requirement ensures that all kinds of generation approaches/implementations can collaboratively generate meaningful actionable workflows. *The ASCII requirement make it easy to generate, parse and extend partial workflows. By requiring order-independence of components we achieve:* *1. The structure of the workflow has to be made explicit inside the components.* *2. generation mechanisms can extend partial workflows at arbitrary open ends, without concern for breaking the structure of the WF.*
	WR.3	**Compactness**
		A WF language shall only require specification of essential components and their essential attributes.
		This requirement ensures that the language is able to represent all necessary components and details of a workflow. Other components and details may be specified by the WF language, but these other components are (must be) optional.
	WR.4	**Compositionality**
		The WF language shall provide facilities to specify a unique identifier in each workflow as well as constructs to defer specification of WF elements to another workflow (the *sub*-workflow) by referring to that sub-workflow's unique identifier.
		This requirement ensures that the language is able to uniquely identify each workflow, and allow cross-referencing from one workflow to another. The cross-reference takes the place of another component, i.e. a reference to a workflow (the sub-workflow) instead of an action.

	WR.5	**Open semantics**
		The WF language shall provide components that can be associated to components of any other type in the WF and that may contain arbitrary (text-)values .
		This requirement ensures that the workflows can contain components that specify domain-specific or activity-specific information. The syntax of these elements is not defined by the syntax specification of the WF language. The purpose of this requirement is that the WF generation services of an agency can include (proprietary or bilaterally negotiated) details about the specific execution/instantiation of a task that can (only) be understood by designated other services.
	WR.6	**Extensibility**
		The WF language shall provide constructs that extend the WF syntax with optional components, elements and values.
		This requirement ensures that the language can be augmented with constructs for specific management duties. A commonly used approach to this augmentation is the 'include' or 'uses' primitive. As an example, syntax-modules containing specification of policies or quality of service definitions may be used to augment the syntax on demand.

4.2 Workflow Languages

In this section we will discuss a number of WF languages that can be considered candidates for use in a Productive4.0 demonstration. The discussion is limited to the major contenders in the domain of workflow representation and the candidates that the Productive4.0 partners are familiar with. Therefore, this chapter discusses BPMN, BPEL, BRAWL, Taverna and YAWL.

4.2.1 **BPMN**

Overview

Business Process Model and Notation (BPMN) is a standard for business process modelling that provides a graphical notation for specifying business processes in a Business Process Diagram. BPMN 2.0 was released in 2011. Since 2014, BPMN has been complemented by the Decision Model and Notation standard, a standard for building decision models.
The objective of BPMN is to support business process management, supporting *only* the concepts of modeling applicable to business processes. Other types of modeling are out of scope for BPMN.
By providing a notation that is intuitive to business users, yet able to represent complex process semantics, BPMN serves both business and technical users. The BPMN specification provides a mapping between the graphics of the notation and the underlying constructs of execution languages, particularly Business Process Execution Language (BPEL).

BPMN is provided in tools such as ARIS Express and Microsoft Visio. Both have their own proprietary format for storage and exchange.

Features

Components

BPMN models consist of diagrams constructed from elements of four basic element categories:

1. **Flow objects**
2. **Connecting objects**
3. **Swim lanes**
4. **Artifacts**

Flow objects

Flow objects are the main describing elements within BPMN, and consist of three core elements: events, activities, and gateways.

Figure 19: BPMN Events

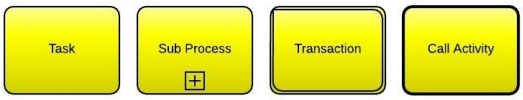


Figure 20: BPMN Activities

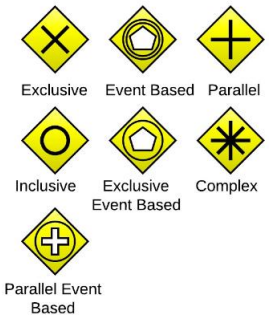


Figure 21: BPMN Gateways

Event

An Event is represented with a circle and denotes something that *happens* (compared with an activity, which denotes what is *done*). Icons within the circle denote the type of event (e.g., an envelope representing a message, or a clock representing time). Events are also classified as catching (for example, if catching an incoming message starts a process) or throwing (such as throwing a completion message when a process ends).

- **Start event**: Acts as a process trigger; indicated by a single narrow border, and can only be *Catch*, so is shown with an open (outline) icon.
- **Intermediate event**: Represents something that happens between the start and end events; is indicated by a double border, and can *Throw* or *Catch*. For example, a task could flow to an event that throws a message across to another pool, where a subsequent event waits to catch the response before continuing.
- **End event**: Represents the result of a process; indicated by a single thick or bold border, and can only *Throw*, so is shown with a solid icon.

Activity

An activity is represented with a rounded-corner rectangle and describes the kind of work which must be done. An activity can be atomic or compound.

- **Task**: A task represents a single atomic unit of work. A task is the lowest level activity illustrated on a process diagram. A set of tasks may represent a high-level procedure.
- **Sub-process**: When collapsed, a sub-process is indicated by a plus sign against the bottom line of the rectangle; when expanded, the rounded rectangle expands to show all constituents. A sub-process is referred to as a compound activity. It has its own start and end events; sequence flows from the *parent* process must not cross the boundary.

- **Transaction**: A form of sub-process in which all contained activities must be completed to meet an objective, and if any one of them fails, they must all be compensated (undone). Transactions are differentiated from expanded sub-processes by being surrounded by a double border.
- **Call Activity**: A point in the process where a global process or a global task is reused. A call activity is differentiated from other activity types by a bolded border around the activity area.

Gateway

A gateway is represented with a diamond shape and determines forking or merging of flows, depending on the conditions expressed:

- **Exclusive**: Creates alternative flows in a process: only one of the paths can be taken.
- **Event Based**: The selected flow is based on an evaluated event.
- **Parallel**: Used to create parallel flows without evaluating any conditions.
- **Inclusive**: Used to create alternative flows where all inflows are evaluated.
- **Exclusive Event Based**: creates alternative flow based on evaluated event.
- **Complex**: Used to model complex synchronization behavior.
- **Parallel Event Based**: Two parallel flows are started based on an unevaluated event.

Connecting objects

Flow objects are connected to each other using **Connecting objects**, which are of three types: sequences, messages, and associations.

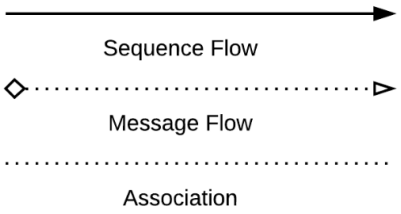


Figure 22: BPMN Connections

- A **Sequence Flow** denotes the order of activities. Solid line and arrowhead.
- A **Message Flow**: A message flow denotes messages flowing *between pools*. Dashed line, open circle at the start, and open arrowhead.
- An **Association** is used to associate an artifact to a flow object. Dotted line, optional open arrowhead.

Swim Lanes

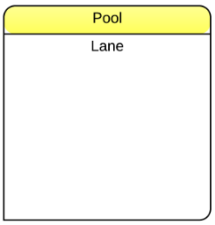


Figure 23: BPMN Swim Lanes

Swim lanes organize activities, based on cross-functional flowcharting. There are two types, pool and lane.

- **Pool:** A pool represents major participants in a process, typically separating different organizations. A pool contains one or more lanes. A pool can be open (i.e., showing

internal detail) when it is depicted as a large rectangle showing one or more lanes, or collapsed (i.e., hiding internal detail) when it is depicted as an empty rectangle stretching the width or height of the diagram.
- **Lane**: A lane is used to organize and categorize activities within a pool according to function or role, and depicted as a rectangle stretching the width or height of the pool. A lane contains the flow objects, connecting objects and artifacts.

Artifacts

Artifacts allow to bring some more information into the model/diagram. In this way the model/diagram becomes more readable. There are three pre-defined artifacts: data object, group and annotation.

Figure 24: BPMN Data object

Figure 25: BPMN Group

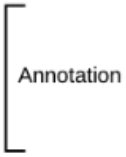


Figure 26: BPMN Annotation

- **Data object**: Data objects show the reader which data is required or produced in an activity.
- **Group**: A Group is represented with a rounded-corner rectangle and dashed lines. The group is used to group different activities but does not affect the flow in the diagram.
- **Annotation**: An annotation is used to give the reader of the model/diagram an understandable impression.

Example of business process diagram

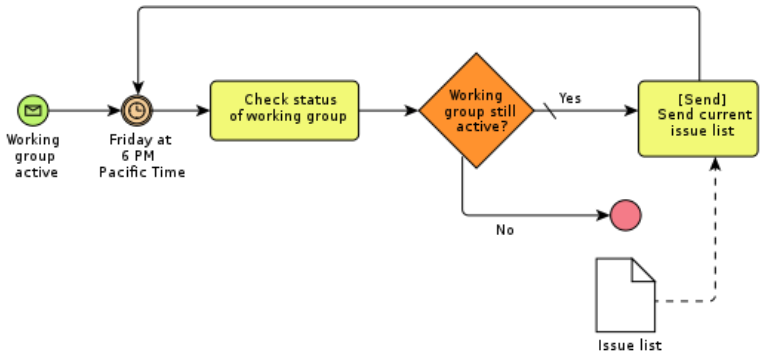


Figure 27: example BPM

Figure 27 presents a sample BPMN diagram, describing the business process where an issue list is sent every week, as long as some working group is active.

Evaluation of Requirements

In this section we evaluate the features of BPMN to the requirements in section 4.1.1.

WR.1 Fit for collaborative context	
	BPMN does not provide a location for generation and management information
WR.2 Fit for Generation	
	BPMN does not have an ASCII-based representation format, although providers of tools supporting BPMN might have such a format available.
WR.3 Compactness	
	As BPMN is graphics oriented, the representation is not compact
WR.4 Compositionality	
	BPMN possesses features for compositionality
WR.5 Open Semantics	
	BPMN provides a free format annotation structure
WR.6 Extensibility	
	Although BPMN appears to be extensible, it is unclear how this is supported by BPMN, and moreover, whether extensions are accepted/supported between various BPMN tooling environments

Overall, the score of BPMN for collaborative WFGM purposes is:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
BPMN	--	--	--	+	+	0

4.2.2 BPEL

Overview

BPEL is a language, standardized by OASIS, to model the behaviour of *executable* and *abstract* business processes. An executable business process models the actual behaviour of a participant in a business interaction. An abstract business process is a partially specified business process, not intended for execution. BPEL, or WS-BPEL, originates from languages such as the Web Services Flow Language (WFSL, [77]); this implies web services are a founding concept in BPEL.
BPEL extends the web service interaction model with support for business transactions. In addition, it defines an interoperable integration model facilitating the expansion of automated process integration within and between businesses. Interactions are shaped as web service operations using WSDL [76]. Business processes are defined using XML schemes.

There is no standard graphical notation for WS-BPEL, but some have proposed to use a the BPMN as a graphical front-end to capture BPEL process descriptions. As an illustration of the feasibility of this approach, the BPMN specification includes an informal and partial mapping from BPMN to BPEL 1.1. However, the development of tools has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models.

Features

BPEL is an XML based language that deploys standards such as WSDL and XPATH [78] to specify elements.
BPEL uses WSDL to define incoming and outgoing messages. XPATH (and XSLT [79]) are used for specifying queries and data retrieval operations.

It contains programming constructs that allow the specification of queries, flow-control and event handling. BPEL also provides data manipulation functions for the simple manipulation of data needed to define process data and control flow.

BPEL also features

- Serialized scopes to control concurrent access to variables.
- Control-specific activity types, such as repeatUntil, forEach, validate, rethrow,
- XSLT for variable transformations
- Locally declared messageExchange (internal correlation of receive and reply activities)

Evaluation

WR.1 Fit for collaborative context	
	BPEL provides a standardized syntax and semantics for workflows in collaborative contexts. It does not contain specific facilities to represent WFGM oriented metadata or represent partial workflows
WR.2 Fit for Generation	
	BPEL is an XML-based language. However flows are incorporated in the structure of the representation, which hampers generation.
WR.3 Compactness	
	BPEL is a rather large language, featuring many specialist details. It may be possible to select a core set of constructs to be accepted by all stakeholders
WR.4 Compositionality	
	Compositionality is not a native construct in BPEL
WR.5 Open Semantics	
	BPEL aims to be as complete as possible, specifying all constructs, elements and values that were relevant at the time of definition. This does not allow non-standard or proprietary elements to be included. BPEL does feature elements “attribute” and “document” that may contain user defined values.
WR.6 Extensibility	
	The development and specification of BPEL is governed by OASIS.

Overall, the score of BPEL for collaborative WFGM purposes can be summarized as:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
BPEL	-	+	+	-	0	-

4.2.3 **BRAWL**

Overview

The BRIDGE Annotated Workflow Language (BRAWL) is a workflow representation language, developed in the FP7 BRIDGE project, and is targeted to support the entire process of workflow generation and management, including quality of service management.

BRAWL defines syntax and semantics of workflow elements. BRAWL allows the specification of incomplete workflows (*partial workflows*), which can be further completed (that is: extended) by any of the available workflow generation systems. The BRAWL language is positioned as a vehicle for communication and collaborative workflow management.

BRAWL is used by WFGM systems such as ATOM, COMPASS/SMDS and CoWS (see sections 5.2.1, 5.2.3 and 0 respectively). The specification of the current version of BRAWL is included in this deliverable in Appendix 9.2.

Features

Structure

BRAWL is an XML-based language that specifies expressions with two groups: a *banner*, and a body.
The banner identifies the expression as a BRAWL workflow representation, and contains the BRAWL version number.
The body has a unique identifier and is subdivided in a *preamble* and a *workflow*. The identifier is used to refer to from other (sub-) workflows. The preamble contains relevant information for generation and management services that process the workflow representation.

Core Components

The workflow part of a BRAWL workflow contains the actual representation of the workflow. In its simplest form, the workflow is composed of the BRAWL key concepts:

- Actions, describing tasks that need to be executed
- Artefacts, describing results from Actions
- Arrows, connecting Actions and Artefacts
- Resources, describing entities that are able to execute Actions.
- Annotations, describing relevant properties of Actions, Artefacts, Arrows and Resources

Annotations may include values that cannot be understood by all workflow interpreters. The principle is that the statements containing these values are ignored and, importantly, left unmodified by the interpreter. These statements will be processed by other management processes that can understand these values. The principle of ‘leave the stuff you don’t understand alone’ provides us with the ability to enforce a clean separation of concerns between the management services.

Values

The values of elements of core components can be 'plain', such as a number of a text-string, or take the form of evaluable expressions, such as (simple) arithmetic or logic expressions, or even scripts.

Extensions

BRAWL is set up to be modular, implying that new modules containing new keywords for special concerns can be included. So, the basic version of BRAWL includes all the keywords and constructs required to specify annotated workflows, scripts and references to other workflows and modules. The BRAWL modules are intended to contain the tokens and keywords for specific issues, for example, trust, policies or security.

Evaluation

WR.1 Fit for collaborative context	
	BRAWL was created for collaborative deployment. BRAWL provides structures and elements that capture information relevant during a collaborative effort, such as generator identities, current relevant global values for workflows. Using Annotations, BRAWL allows inclusion of arbitrary operational information in a workflow, such as: • Quality criteria and conditions • Service Level Agreements • Resource allocations • Configuration data and recipes BRAWL allows the representation of partial workflows, where the open ends in the partial workflows are identified in the preamble.
WR.2 Fit for Generation	
	BRAWL is fully ASCII based; workflow elements are order-independent
WR.3 Compactness	
	BRAWL features a very compact core, but allows generation and management services to make use of dedicated formats in annotation values and various (predefined) extensions to express domain-dependent information.
WR.4 Compositionality	
	BRAWL provides features to create compositional workflows.
WR.5 Open Semantics	
	The Annotation component in BRAWL can be associated to any other (type of) component and can contain any value desired in ASCII representation. The interpretation of annotations is left to the management services. The principle for interpretation is to leave elements, which cannot be processed by some interpreter, alone and preserve them for interpreters of service that can make use of these elements.
WR.6 Extensibility	
	BRAWL provides the syntax to define and use extensions. Among the current extensions are modules containing language constructs for Service Level Agreements and QoS management

Overall, BRAWL scores rather well, which is unsurprising as BRAWL was designed to support the exact requirements stated in section 4.1.1:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
BRAWL	++	+	++	+	+	+

4.2.4 Taverna

Overview

Taverna, also Apache Taverna, is a workflow design and execution tool, combining a graphical workflow design environment, creating workflows using an XML-based workflow representation language. Taverna originates from the myGrid project, but is now a project under the Apache Incubator program. Taverna allows the inclusion and integration of components based on SOAP/WSDL or REST web services.
Taverna is open source and features its own shared on-line workflow experimentation and execution environment, myExperiment.

For the purpose of this deliverable, information on Taverna was obtained from Wikipedia [https://en.wikipedia.org/wiki/Apache_Taverna], the Apache Incubator website, [https://taverna.incubator.apache.org/], as well as the Taverna documentation that can be downloaded from the project homepage on the Apache Incubator site.

Overview

Taverna workflows are designed and created in an integrated workflow development environment, the Taverna Workbench. The workflows created by Taverna can invoke general SOAP/WSDL or REST Web services, and can also invoke R statistical services, local Java code, external tools on local and remote machines (via ssh), do XPath and other text manipulation, import a spreadsheet and include sub-workflows.

Taverna also includes the capability to search for workflows on myExperiment. The Taverna Workbench can download, modify and run workflows discovered on myExperiment, and also upload created workflows in order to share them with others on myExperiment.

Taverna workflows do not need to be executed within the Taverna Workbench. Workflows can also be run by:

- a command line execution tool
- remote execution server that allow Taverna workflows to be run on other machines, on computational grids, clouds, from Web pages and portals
- online workflow designer and enactor OnlineHPC

Taverna allows pipelining and streaming of data. Taverna workflows are primarily data-driven rather than control-driven. Workflow components can possess an arbitrary number in- and outputs, which have to be defined and connected by the creator of Taverna workflow. Taverna specifies activity elements. Connections are included in the activity elements Taverna possesses the notion of annotations but these are included in the elements rather than represented as standalone elements.

Taverna workflows are represented in an XSM-based language, which is hidden by the graphical tool bench. The user has no direct control over this representation; this prevents errors in the XML representation, but hampers an automated collaborative generation

Evaluation

WR.1 Fit for collaborative context	
	Taverna is suitable for human collaboration using the facilities in the Taverna Workbench. However automated collaboration seems not to be one of the features of Taverna. As such Taverna has no placeholders in the workflow representation specification to host WFGM information.
WR.2 Fit for Generation	
	Taverna is an XML-based language. However, the representation locates connections and annotations inside an activity element, which hampers the creation of partial workflows.
WR.3 Compactness	
	Taverna is compact, although more sophisticated details may be included in the representation.
WR.4 Compositionality	
	Taverna allow compositionality
WR.5 Open Semantics	
	Taverna allows the inclusion of annotations, but the annotations are included in the

	(activity) elements specifications…
WR.6 Extensibility	
	Taverna allows the inclusion of arbitrary scripts and services in the workflow, but does not have a mechanism to include new syntax elements. The syntax of Taverna is controlled by the Apache Taverna project group.

Overall, the score of Taverna for collaborative WFGM purposes is:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
Taverna	-	+	++	+	0	-

4.2.5 YAWL

Overview

Like Taverna, YAWL provides a graphical workflow creation environment that produces XML-based workflow representations. YAWL and its tool bench were developed by researchers at Eindhoven and Queensland Universities of Technology, but also contain contributions from industry.

YAWL is intended to support all of the workflow patterns and have a formal semantics, using the paradigm of Petri nets as a foundation. YAWL extends this paradigm with constructs and syntax elements that allow the creation of workflows that are not directly supported in Petri-nets. The formal semantics of YAWL has created the opportunity to develop static analysis tools for processes, WofYAWL.

YAWL Provides support for workflow patterns, which are distinguished in four perspectives:

1. control-flow
2. data
3. resource
4. exception handling

There are multiple workflow patterns defined for each of the perspectives. Each pattern defines and specifies a composition of workflow components, allowing a comprehensive approach to modular workflow design, to some extent yielding predictable qualities and behaviour. Evaluating a pattern's requirements and constraints helps ascertain whether a workflow system supports the pattern.

For the purpose of this deliverable information on YAWL has been obtained from the Wikipedia [https://en.wikipedia.org/wiki/YAWL] and the developer's website [http://www.yawlfoundation.org/], as well as the YAWL documentation that can be downloaded from the developer's website.

Features

- Comprehensive support for workflow patterns.
- Support for advanced resource allocation policies, including four-eyes principle and chained execution.
- Support for dynamic adaptation of workflow models through the notion of worklets.
- Sophisticated workflow model validation features (e.g. deadlock detection at design-time).
- XML-based model for data definition and manipulation based on XML Schema, XPath and XQuery.

- XML-based interfaces for monitoring and controlling workflow instances and for accessing execution logs.
- XML-based plug-in interfaces for connecting third-party web services with the system, including third-party worklist/task handlers.
- Automated form generation from XML schema.

Evaluation

WR.1 Fit for collaborative context	
	YAWL does not contain features for collaborative workflow generation
WR.2 Fit for Generation	
	YAWL possesses an XML-based representation. However, connections are specified inside tasks, which hampers the generation of partial workflows.
WR.3 Compactness	
	YAWL is reasonably compact, however, due to its associated graphical work bench, the workflow representation contains layout information
WR.4 Compositionality	
	YAWL allows the plugging in of third party worklists and task handlers. However compositionality is not a notion in YAWL
WR.5 Open Semantics	
	Due to the principle of formal semantics, YAWL does not have an open semantics; it does not allow contents outside the specification
WR.6 Extensibility	
	The syntax and semantics of YAWL are governed by the YAWL foundation. YAWL does not possess a mechanism to dynamically include new syntax constructs.

Overall, the score of YAWL for collaborative WFGM purposes can be summarized as:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
YAWL	-	+	0	-	-	-

4.3 Conclusion Workflow representation languages

Surveying the collected evaluation of discussed workflow languages in sections 4.2.1 through 4.2.5 yields:

	WR.1	WR.2	WR.3	WR.4	WR.5	WR.6
BPMN	--	--	--	+	+	0
BPEL	-	+	+	-	0	-
BRAWL	++	+	++	+	+	+
Taverna	-	+	++	+	0	-
YAWL	-	+	0	-	-	-

It is evident that BRAWL scores superior to the other candidates; as noted before this is not surprising as BRAWL was designed with the specific goal of collaborative workflow generation and management in mind, as stated in the requirements. The other candidates were designed with other concerns and goals, which has led to different design choices, which may prohibit or hamper the dynamic multi-party collaboration we envision for the Productive4.0 project.

5. Workflow Generation and Management: State of the Art

In chapter 2 we have presented the baseline principles of Workflows Generation and Management, based on a shared workflow representation. As documented in chapter 2, Workflow Generation & Management (WFGM) and Quality of Service Management (QoSM), involve a number of activities that exchange a representation of a (partial) workflow. Therefore, we need to select a WFGM concept that satisfies requirements of collaboration, the activities in WFGM and QoSM.

In this chapter we will list the requirements on WFGM systems in section 05.2. We will then discuss a number of candidate WFGM systems in section 5.2.

5.1 Requirements on Workflow Generation and Management

In chapter 2, we presented the principal process of workflow generation and management, the graphical representation of this principle is repeated in Figure 28.

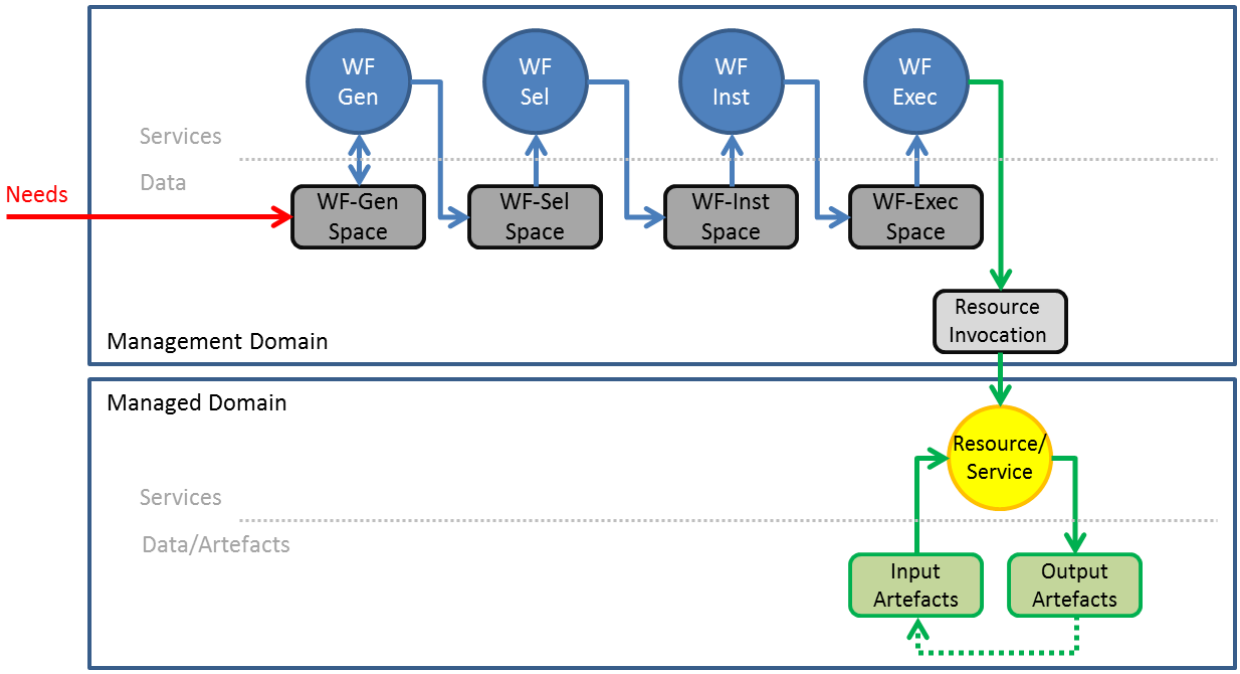


Figure 28: Workflow Generation & Management principle

Also in chapter 2, we discussed some necessary characteristics of collaborative WFGM. In résumé of this discussion, a collaborative WFGM system features:

1. The ability to perform in collaboration with peer services, allowing these peer services to provide their contributions using an iterative, shared and coordinated operation
2. The individual services in the system are required to be open, allowing each service to deal with the information in (partial) workflows that is outside the scope of its expertise.
3. A complete set of capabilities, ensuring that, if adequate workflows can be produced, these workflows *will be* produced; in addition, the system should possess the capabilities to select, instantiate and execute valid workflows.

These characteristics are laid down in the requirements on WFGM systems for Productive4.0.

5.1.1 Requirements elicitation

Workflow Generation and Management system Requirements		
	WG.1	**Collaboration**
		A WFGM system shall be usable in a collaborative setting, meaning that the system generates workflows in a shared workflow language in an iterative fashion.
		This requirement implies that candidate systems must be designed to generate and manage workflows in collaboration with other systems, originating from other developers. the WFGM system of systems will standardize on output, which means requirements on the partial workflows produced in terms of syntax and semantics need to be met by each of the contenders.
	WG.2	**Openness**
		A WFGM system shall be open to process workflow statements generated to specification by other partners, and tolerant for expression element that are outside the domain of expertise of an individual service.
		In a multi-disciplinary BPM environment, not all the domain knowledge will reside with one partner: for considerations of integrity, autonomy and strategy, knowledge will be applied in workflows in line with the expertise of each stakeholder. This will, in general result in workflow representations that contain information that cannot be understood by all service providers. A service provider that encounters elements in a workflow that it cannot process, must: *1. not fail on the incomprehensible element* *2. not modify or tamper with the incomprehensible element* *3. not remove or discard the incomprehensible element.*
	WG.3	**Completeness**
		A WFGM system shall provide a complete set of services for workflow generation and management, specifically services for generation, selection, instantiation and execution of workflows.
		This requirement ensures that a candidate system contains all the functionality to perform WFGM responsibilities. *However, some concepts provide workflow generation services, but not, for example instantiation services. In case we want to use this candidate anyway, additional providers for the lacking services need to be found; this is specified in the relaxed requirement WG.3r:*
	WG.3r	**Partial Completeness**
		In case a WFGM system provides just part of the required services generation, selection, instantiation and execution, it must be extended with services (from other providers) that cover the lacking capabilities.
		This requirement states that it must be possible to achieve a complete (and compatible) set of services in case only some of them are provided by a candidate.

5.2 Workflow Generation and Management systems

In this section we will discuss a number of WFGM systems that can be considered candidates for use in a Productive4.0 demonstration. The discussion is limited to the major contenders in

the domain of WFGM and the candidates that the Productive4.0 partners are familiar with. Therefore, this chapter discusses Arrowhead, Atom, COMPASS/SMDS Cows, Taverna and YAWL.

5.2.1 Arrowhead

Arrowhead is a framework for service registry and orchestration, aiming to enforce the Service Oriented Architecture (SOA) principles Lookup, Loose coupling and Late binding for systems of systems. It projects three core service systems in any compliant network: the Information Infrastructure (II), the Information Assurance (IA) and the System Management (SM).

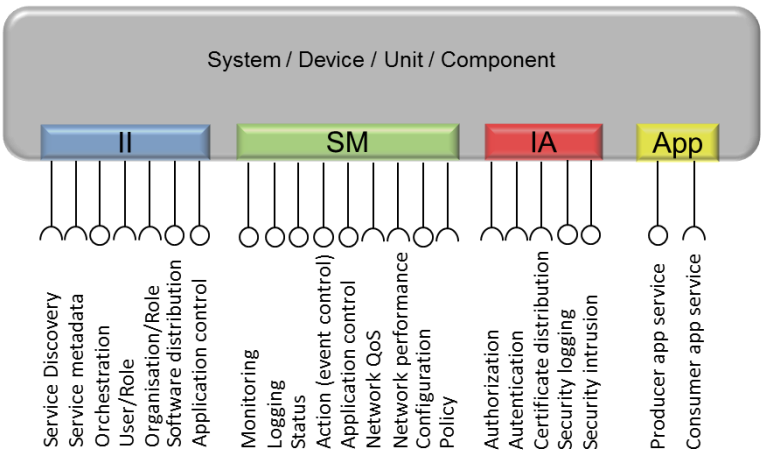


Figure 29. An Arrowhead component with facilities for all possible II, IA and SM services

The operational principle of Arrowhead is to collect service descriptions in a service registry and perform orchestration by instructing service providers what consumers to accept and service consumers what services to consume. The service transactions themselves take place without the intervention of Arrowhead services.

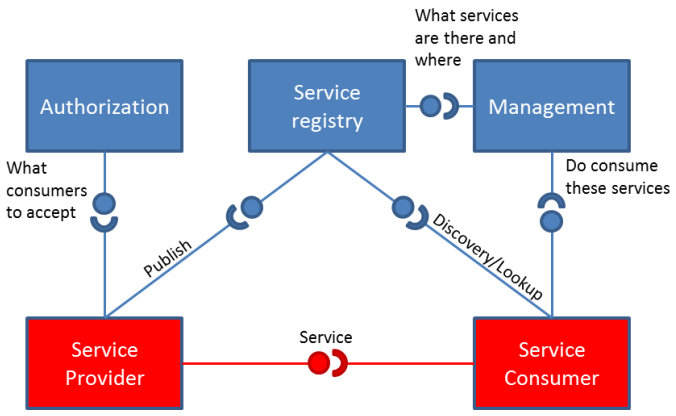


Figure 30. Arrowhead operational principle

Arrowhead does not provide a design principle for the orchestration service itself, but rather provides protocols and interfaces to perform orchestration. Orchestration is performed at the lowest level, and the notion of workflows is not present in the design. A specific workflow can however be enforced using a priority mechanism, and provided the workflow is generated

elsewhere, the orchestration services can be deployed to configure the service providers and consumers.

Arrowhead is used in a collaborative setting, but as such does not provide workflow generation or selection engines. It does provide instantiation and execution services, which could possibly be extended with generation and selection services from other sources. It is unclear whether Arrowhead can be configured to operate in collaboration with non-Arrowhead systems.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
Arrowhead	+	0	-	+

5.2.2 ATOM

The Agent-based Team Organization Methodology (ATOM) is based on a workflow concept that combines bottom-up task negotiation with top-down goal assignment. The management services take the form of agents that perform actions in a workflow from the perceived current situation towards a goal, the desired end-situation.
The structure of an agent resembles a control systems feedback loop (see Figure 31), and the actions it places in short-term workflows are delegated to operational resources.

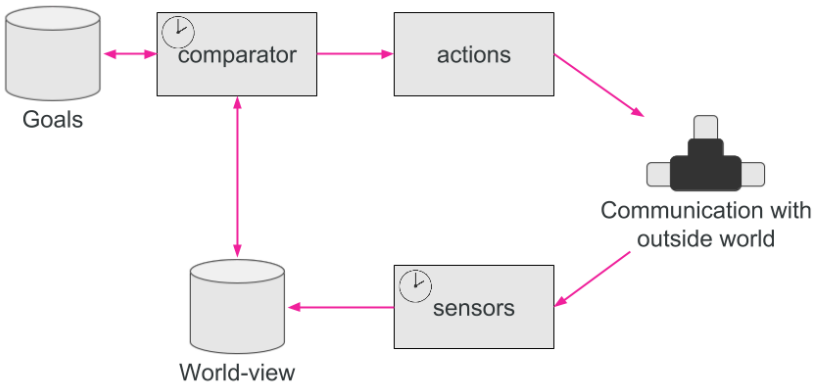


Figure 31: ATOM Feedback Loop

- Actions change to the outside world: the actions result from assigning tasks to operational resources
- Sensors monitor the effect of assigned tasks in the outside world: the status of tasks and teams, and other relevant contextual information gained
- Observations from sensors are used to update the World-view, reflecting the current situation and the status of tasks, teams, resources, etc.
- A comparator compares the perceived situation (world-view) with the desired situation based on the Goals;
- Differences in perceived and desired situation result in new actions until the goals have been achieved.

By grouping agents in teams, agents can coordinate progress towards the desired situation, where the teams keep track of the combined effect of the actions issued by the individual agents.

New goals are assigned to a team, represented by a dedicated team-agent, and each agent determines whether and how it can effectively participate in achieving the goal. This (possible) contribution is communicated to the team-agent, which selects the candidates with the best propositions for actions and rejects the gents with lesser propositions; this mechanism is called dynamic team formation.

The team-agent effectively combines the roles of workflow generator and selector, building workflows from the proposed (partial) workflows. The (other) operational agents perform the role of instantiation and execution.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
ATOM	+	-	+	=

5.2.3 **COMPASS/SMDS**

The Thales Self-Managing Distributed Systems (SMDS) concept is based upon the delegation of planning, execution, monitoring and federation responsibilities to dedicated system services. COMPASS (Configuration, Organisation and Management Prototype for Autonomous Systems of Systems) is an implementation prototype of SMDS. For an overview of the prototype, see Figure 32. All planning activities are executed in the Planning (P) service, which generates workflows based on Needs from clients or stakeholders. A Need is a formulation of *initial requirements*. The Planning service produces an executable workflow, which is used by the Instantiation (I) service to determine an allocation of the tasks and roles in the planning to the available resources, called the *managed subsystems*.

This allocation is based on a topology description provided by the Federation (F) service and status data provided by the QoS Monitoring (Q) service. The QoS Monitoring segment monitors and evaluates the progress of the executed workflow and compares this progress with the initial requirements as stated in the Needs. The QoS Monitoring service provides *status information* to the Instantiation service for load balancing and allocation adjustment purposes, a*djustment triggers* to the Planning service in case the workflow needs to be adjusted and *escalation triggers* to the Federation service, to request additional resources in case the initial requirement cannot be satisfied. The Instantiation service allocates tasks to resources in the managed subsystems; the resources in the managed subsystems are contributed by the stakeholders and described in the topology description provided by the Federation service. The Federation service joins and separates the platforms containing resources in the collaborative system, collects information about the contributed resources (in the joined platforms), federates WFGM and QoS Management servers into service networks and provides a topology description to the Instantiation service. The Federation service can also ask for escalation of the system of systems (increase the number of resources) or de-escalation (decrease the number of resources).

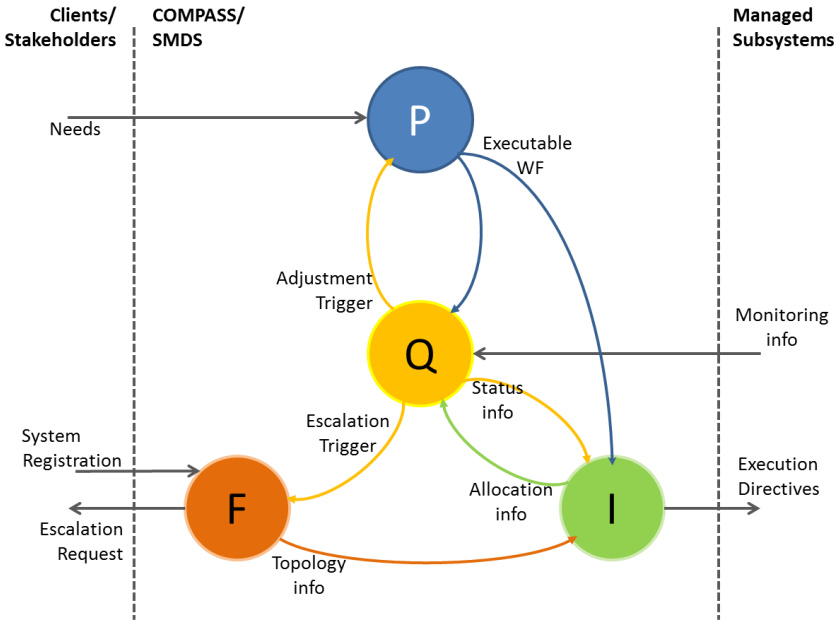


Figure 32: COMPASS/SMDS overview

These four service segments of COMPASS/SMDS form the self-management capability of a self-managing dynamic collaboration system. In combination the mechanisms of these segments coordinate the behaviour of the system, by instantiating and terminating processes, allocating tasks to processes and interconnecting processes. In Figure 32 the service and the information these exchange are depicted. In Figure 32, the Planning service is represented by the blue circle 'P', the QoS Monitoring service by the yellow circle 'Q', the Instantiation service by the green circle 'I' and the Federation service by the orange-brown circle 'F'.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
COMPASS/SMDS	+	+	+	=

5.2.4 CoWS

CoWS [20] is a top-down, template-based approach to automated reconfiguration of web services. CoWS aims at automated adaptation of complex services, in case a need for adaptation emerges, for example, when services become unavailable. CoWS enhances the possibilities for reconfiguration beyond the simple adaptation strategies available in most current approaches, which are often limited to supporting replacement of black-boxed services or pre-determined fixes.

CoWS uses templates to express local knowledge, structure complex services, and provide a means to represent configurations of services and knowledge about the configuration. Both templates and web services are used to structure a composition, combining them to form a template-based web service configuration. Both templates and web services have properties, which take the form of annotations. To iteratively create a configuration, templates have slots, which impose requirements that can be matched with the properties of web services and templates.

A *template* describes a single-level composition of web services. A template is defined as a control structure with one or more slots, an associated template description, and dependencies between slots.

A *slot* is a placeholder, defining the requirements for a desired (single) service or template. A requirement is an explicit expression about qualities of a function, behaviour or structure. A *control structure* defines the conditions for activation of the slots. These are specified using control constructs over a set of slots. Examples of control constructs are sequential/parallel activation and conditional constructs (if-then-else).

The template description specifies the properties of the template and the associated values, describing the function and behaviour. The template can contain additional dependencies.

The process of creating a workflow, see Figure 33, contains the tasks:

- Focus Determination: Determines the part of the initial configuration for which a replacement is to be created, setting the scope of reconfiguration.
- Requirement Determination: Determines the requirements applicable to the focus.
- Template-based Configuration: Creates a new configuration, satisfying the requirements that can replace the part in focus.
- Integration: Removes the failing part of the configuration and replaces it with the created configuration, resulting in a new web service configuration.

The following information items are used by the reconfiguration process:

- Initial configuration: The template-based web service configuration to be reconfigured.
- Failing web service: A pointer to a service in the initial service configuration that needs to be replaced.
- Repositories: A pointer to repositories containing web services and templates that can be used by the reconfiguration process.
- Focus: A pointer to a slot containing a service or a template in the initial service configuration, representing the part for which the reconfiguration process is currently determining a replacement. If the value of a focus is null, this implies that no focus could be determined.
- Requirement set: A collection of requirements, related to a specific focus.
- Created configuration: A template-based web service configuration that satisfies the determined requirement set, created by the configuration process. If the value of the created configuration is null, this implies that no web service configuration could be found that satisfies the given requirement set, based on the available repositories.
- New configuration: The resulting configuration that will act as a replacement for the initial service configuration, without the failing web service. This configuration is a combination of the initial configuration with the created configuration.

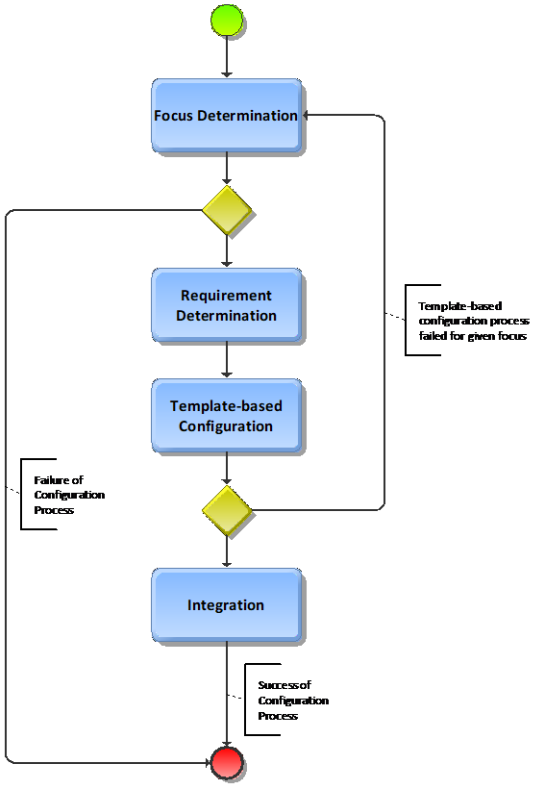


Figure 33: Process representation of CoWS configuration process

Although it is somewhat counterintuitive to bootstrap CoWS to initiate workflow generation from an empty workflow, CoWS has been proven to work within a collaborative workflow generation system. CoWS features generation and selection capabilities, but delegates instantiation and execution to other services.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
CoWS	+	+	-	+

5.2.5 Taverna

Runtime is for in-silico simulation of scientific workflow execution, intended for web services (WSDL style), incubated at apache.org. As such Taverna workflows are not generated but designed and manually entered using the Taverna Toolbench, see Figure 34.

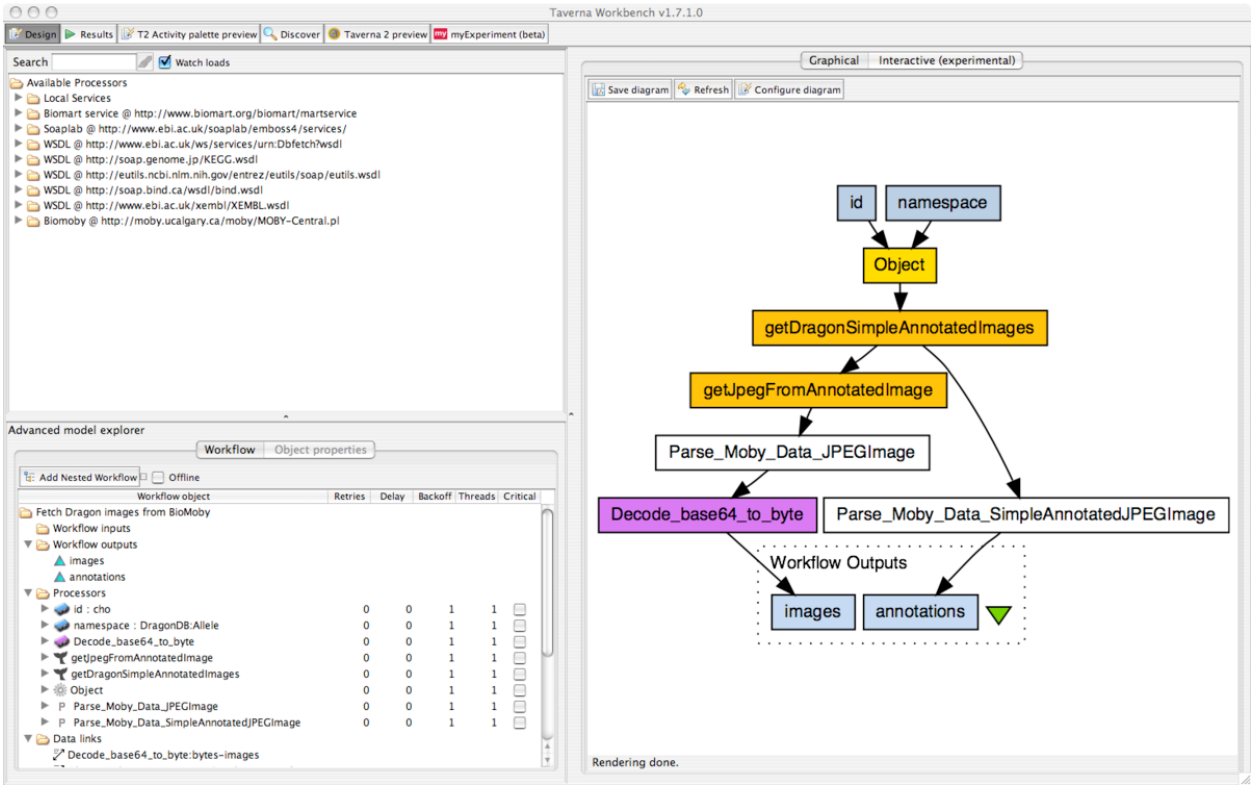


Figure 34: Taverna Workbench

Taverna is able to execute workflows, provided the allocated resources are compatible with the Taverna system. Although Taverna is unfit to operate in an automated collaborative environment, Taverna could be used to include the human brain in the generation and selection process: human operators could design the high-level business process or workflow and delegate the generation of details to the automated generation facilities. Furthermore, the human operator could perform the selection task, selecting the most desirable workflow from a number of candidates.
The premise for this is that either the WFGM facilities generate Taverna workflows, or that a back-end for Taverna, producing workflows in the target language, can be implemented.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
Taverna	-	-	-	+?

5.2.6 YAWL

The situation for YAWL is similar to the situation of Taverna: It contains a graphical user interface to design and validate workflows (Figure 35). Differences with Taverna are that YAWL focusses on analysis. Tasks in workflows can be mapped to “worklets” and human operators.

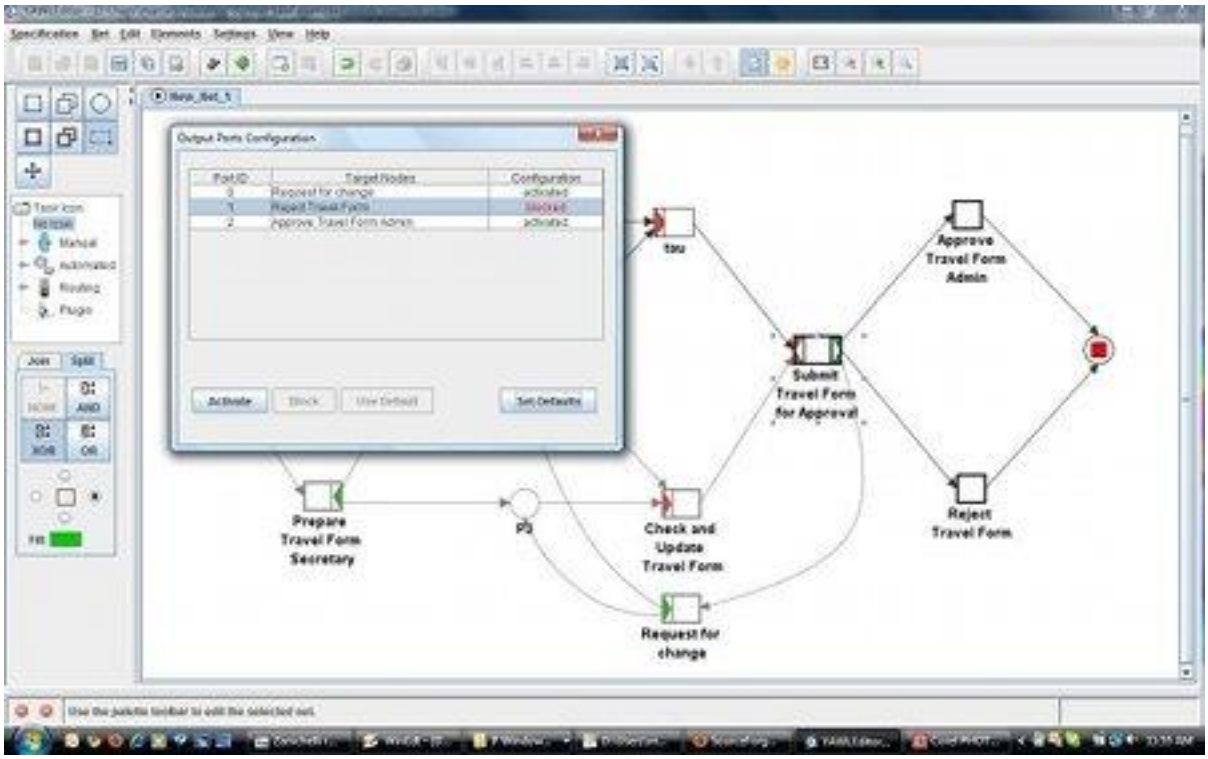


Figure 35: YAWL Toolbench

So, similar to Taverna, YAWL is able to execute workflows, provided the allocated resources are compatible with the YAWL worklets paradigm. Although YAWL is unfit to operate in an automated collaborative environment, YAWL could be used to include the human brain in the generation and selection process: human operators could design the high-level business process or workflow and delegate the generation of details to the automated generation facilities. Furthermore, the human operator could perform the selection task, selecting the most desirable workflow from a number of candidates.
The premise for this is that either the WFGM facilities generate YAWL workflows, or that a back-end for Taverna, producing workflows in the target language, can be implemented.

Evaluation

	WG.1	WG.2	WG.3	WG.3r
Yawl	-	-	-	+?

5.3 Conclusion Workflow Generation and Management

Collecting the evaluation results of the candidate WFGM systems with respect to the WFGM requirements yields:

	WG.1	WG.2	WG.3	WG.3r
Arrowhead	+	0	-	+
ATOM	+	-	+	=
COMPASS/SMDS	+	+	+	=
CoWS	+	+	-	+
Taverna	-	-	-	+?
Yawl	-	-	-	+?

COMPASS/SMDS constitutes a complete suite of WFGM services. ATOM, COMPASS/SMDS and CoWS have been proven to be able to collaborate on workflow generation and management, where COMPASS and CoWS can be tasked with the high-level generation of workflows, while the details of allocation and execution can be delegated to both ATOM and COMPASS.

Since ATOM, COMPASS and CoWS lack a graphical user interface, systems such as Taverna or YAWL could be deployed to formulate a (high level) business process and visualize workflows resulting from automated generation. Automated generation could be the responsibility of systems such as COMAPSS and CoWS, while low level instantiation, execution and orchestration could be facilitated by Arrowhead, ATOM and COMPASS. This train of thought is further explored in chapter 7.

6. Quality of Service Management: State of the Art

In chapter 2 we have presented the baseline principles of Quality of Service Management, based on a shared workflow representation. As documented in chapter 2, Workflow Generation & Management (WFGM) and Quality of Service Management (QoSM), involve a number of activities that exchange a representation of a (partial) workflow. Therefore, we need to select a QoS Management concept that satisfies requirements of collaboration imposed on the activities in WFGM and QoSM.

In this chapter we will discuss and elicitate the requirements on QoSM systems in section 6.15.2. We will then discuss a number of candidate WFGM systems in section 6.2.

6.1 Requirements on QoS Management

To facilitate collaboration and coordination in a Productive4.0 ePLM use-case, we require a adequate approach to Quality of Service management. The QoS management services will play a central role creating reliable, meaningful collaborations, since they will support the definition of the allowable boundaries of operations in the collaboration and, at the same time, enforce these boundaries. Therefore, the QoS management services must support the specification and enforcement of:

1. operational constraints and conditions in a workflow that provide the assurance of operational integrity.
2. system and agency specific rules of engagement that endure the protection of the integrity of the system of systems, the participating agencies and their resources.

These constraints, conditions and rules of engagement are specified in QoS Descriptions, Service Level Agreements and Policies.

6.1.1 QoS Descriptions

A Quality of Service description quantifies the level of a service (to be) provided. This quantification can be on non-functional aspects of the service (like security level) or functional aspects (like throughput, latency or available/negotiated bandwidth for a data transmission service).

A well-specified QoS description can be used for a number of purposes:

1. QoS descriptions can be used to express QoS specifications of resources in a resource or capability repository. In this case a QoS description describes for example the maximum (or minimum) attainable service level for a particular resource.

2. QoS descriptions can be used to formulate conditional clauses in Policies. In this case, the QoS description is interpreted as the condition that fires a policy rule.
3. QoS descriptions can be used in Service Level Agreements to express the expected (agreed upon) service level. Deviations from the expected service level will lead to mitigation. For example, QoS descriptions can be used to parameterize monitoring tasks. In this case the QoS description specifies the conditions under which the monitor is to trigger an alert.

A QoS description relates a service quality to a value or a range of values. This implies that, a simple expression format, containing the quality name, a relational operator a value and a unit would in principle suffice.

Besides the services performing registration and generation tasks (as discussed in section 2.4), the QoS management services require services that can evaluate QoS descriptions, that is, compare the actual (achieved) value of a quality with the required value of that quality. For example, if a Service Level Agreement states that the Bandwidth to be provided for some task shall be at least 5 Megabit per second ("bandwidth >= 5.0 Mb/s"), a monitoring mechanism must be able to sample the provided bandwidth and ascertain that it is actually at least 5 Megabit.

Policies

Policies express expected or intended behaviour under given circumstances. Policies are used to specify:

- *integrity constraints* for the self-protection of agency and resource against undesirable quality requirements;
- *rules of engagement*, specifying condition- and scenario-bound interactions and responsibilities in common situations; (the word "common" reflects expected operational conditions in this case)
- *escalation* and *de-escalation* rules indicating expected activities/behaviour in exceptional situations.

Policies may be defined for all system levels to direct the expected behaviour on that level. We distinguish between policies on the System of Systems-level, the system-level and the resource level.

On the System of Systems level, policies can be used to direct escalation/de-escalation and collaboration configuration and adjustment. For example, an escalation policy can require the System of Systems to attract new agencies and resources in case none of the participating agencies is able to provide the resources or capabilities at a required level.

On the level of individual systems, policies are used to specify the rules of engagement of that system. An example of such a policy is the rule that some activity shall not be performed at a location where the cost of transport exceeds the value of the operation. System level policies are defined by the agency that owns the system. In this deliverable we will reserve the term *rules of engagement* for system level policies. A rule of engagement for a system applies to (is valid for) all components of that system.

On the resource level, policies are used to express integrity constraints of that resource. Integrity constraints include both intrinsic constraints (stating physical, technical or skill-related limitations) of the resource, as well as constraints imposed by the agency that owns the resource, for example legal or ethical constraints. For example, an intrinsic constraint of a resource is the maximum load of that resource. A legal constraint of a resource could be that the resource shall not be deployed in an "unsafe condition", for example, deploying a paint sprayer in a room without ventilation.

Policies can be formulated as expressions that describe a situation in terms of conditions and a (mitigation) action, thereby defining constraints to be satisfied. In a simple form a policy can be represented as an *if-then-where*-rule:

> *If* the following conditions hold, *then* the resources of the organisation are available to be used *where* the following set of constraints holds.

Service Level Agreements

A Service Level Agreement (SLA) specifies a contract in which a party (the supplier) supplies a service at a certain level of quality to another party (the client). A SLA specifies the QoS as the terms and conditions at which the service is offered. Within the boundaries of the agreement, the supplier is free to adapt his planning or provided service-level. Agreements explicate expected interactions of involved parties and may define sanctions if the agreed interactions are not met. SLAs facilitate supporting legal aspects of automated workflow management in the context of crisis management.

A single workflow may contain multiple SLAs, each SLA covering one or more activities. In workflows with automated execution involving multiple organisations, SLAs are commonly used and different frameworks are offered (e.g., WSAS [69], WSLA [70]). Of these two frameworks WSAS is more adapted to open heterogeneous environments, which is more applicable to the Productive4.0 context. SLAs also help human-machine collaboration, as agreements define the boundaries of flexibility between autonomous entities.

A common standard for SLA Agreements is WS-Agreement [69]. In WS-Agreement, the SLA is structured as:

1. Context, containing meta-information, such as
    a. initiator
    b. parties involved
    c. duration of the SLA
2. Terms, composed of
    a. Service description, the functional specification
    b. Guarantee terms, the associated non-functional specifications

QoS Information model

QoS descriptions form the basis for the expression of values and criteria in the decision making processes Quality of Service management services. A QoS Management Information meta-model is presented in Figure 36.

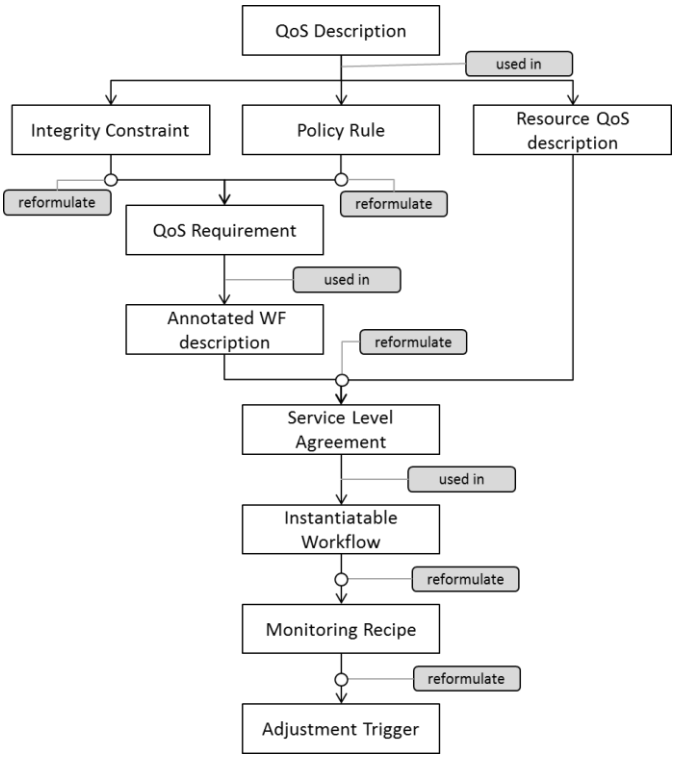


Figure 36: QoS Information Meta-Model

The QoS Information meta-model features two types of arcs: 'used-in'-arcs and 'reformulate'-arcs. The 'used-in'-type arcs denote that the source(-artefact) is used unmodified in the destination(-artefact). The 'reformulate'-arcs denote that contents of the source(-artefact) are reformulated and used in the destination(-artefact). Reformulation is in all cases a straightforward process that is performed by the collaborative WFGM mechanisms.
Each Adjustment Trigger is associated to an alert type, as discussed in section 2.8.2.

6.1.2 Requirements elicitation

Quality of Service Management system Requirements		
	WQ.1	**Collaboration**
		A QoS Management system shall be able to perform tasks in collaboration with other QoS Management systems.
		The principle of collaboration must also uphold for QoS management. This implies a shared framework for collaboration needs to be found or developed.
	WQ.2	**QoS specification**
		A QoS Management system shall perform using an explicit language, expressing relevant Quality of Service aspects.
		This requirement implies that relevant aspects of quality of service requirements can be expressed in the workflow and enforced in its execution. Relevant aspects include: 1. *Operational constraints and conditions* 2. *Service Level Agreements* 3. *Policies*
	WQ.3	**Monitoring**

		A QoS Management system shall be capable of performing relevant monitoring tasks.
		Monitoring provides the basis for systems' situation awareness and decision making on mitigation.
	WQ.4	**Mitigation**
		A QoS Management system shall be capable of performing relevant mitigation tasks.
		Mitigation is the purpose of QoS Management: once deviation from desired situation is detected, the performance of the system needs to be adjusted using mitigation actions.

6.2 QoS Management Approaches and Mechanisms

In this section we will discuss a number of Quality of Service management systems that can be considered candidates for use in a Productive4.0 demonstration. The discussion is limited to the major contenders in the domain of QoS Management and the candidates that the Productive4.0 partners are familiar with. Therefore, this chapter discusses, COMPASS/SMDS, Taverna and YAWL.

6.2.1 COMPASS/SMDS

COMPASS is designed to satisfy the QoS requirements stated in section 6.1.2. COMPASS uses BRAWL and BRAWL extensions to represent QoS Descriptions, Policies and Service Level Agreements. In addition, COMPASS generates monitoring recipes to configure monitoring agents that sample the performance of workflow execution.

The syntax of BRAWL QoS Descriptions, Policies and Service Level Agreements and the COMPASS monitoring recipes are included in sections 9.2.2 through 9.2.5.

Evaluation

	WQ.1	WQ.2	WQ.3	WG.4
COMPASS/SMDS	+	+	+	+

6.2.2 Taverna

In the Taverna Workbench facilities are included to monitor the execution of a workflow and the origin of data produced during execution. It can display the details of a workflow execution as a W3C PROV-O RDF provenance graph. The graph is formulated as ZIP file and includes inputs, outputs, intermediate values and the executed workflow definition.

Taverna does not feature explicit Quality of Service descriptions, policy descriptions or service level agreements. These need to be implemented in the design implicitly by the human workflow designer. Although the performance and results of a workflow execution can be logged and visualized, the debugging and mitigation is delegated to the human user. As stated before, Taverna is not intended to be used in a collaborative setting.

Evaluation

	WQ.1	WQ.2	WQ.3	WG.4
Taverna	-	-	+	-

6.2.3 YAWL

YAWL features a system to catch exceptions generated during workflow execution. In addition YAWL possesses capabilities to enforce resource allocation policies, although these are based on patterns rather than individual constraints. YAWL relies on design time validation, but also features runtime monitoring of workflows, as well as capturing execution logs. Yet, mitigation is not an explicit capability in YAWL, and left mostly to the human user. Other quality of service principles are encoded in the behaviour of the workflow elements, rather than documented explicitly. As stated before, YAWL is not intended to be used in a collaborative setting.

Evaluation

	WQ.1	WQ.2	WQ.3	WG.4
YAWL	-	0	+	+

6.3 Conclusion Quality of Service Management

Collecting the scores of the various candidates results in:

	WQ.1	WQ.2	WQ.3	WG.4
COMPASS/SMDS	+	+	+	+
Taverna	-	-	+	-
YAWL	-	0	+	+

Once again, as COMPASS is designed with the stated requirements from use-case and QoS Management in mind, it scores well in evaluation. The other candidates, Taverna and YAWL address different concerns, which makes them less suitable for the intended application in Productive4.0.

7. Résumé, Final Analysis and Conclusions

7.1 Résumé

In this deliverable we presented the baseline principles for multi-agency systems of systems orchestration using workflows as actionable business processes in chapter 2. We discussed the aspects of workflow representation, automated collaborative workflow generation, instantiation management and Quality of Service management. These principles are to be validated in the context of a Productive4.0 use case, which has been discussed in chapter 3. This chapter also provided the architectural requirements for collaborative WFGM and QoS management in an ePLM context.

In turn, we discussed workflow representation (chapter 4), workflow generation and management (chapter 5) and Quality of Service management (chapter 6), where each chapter

formulated requirements on these aspects and evaluated candidates against these requirements.

7.2 Final Analysis

One of the major concerns in the execution of complex, dynamic, distributed multi-agency business processes is to establish a proper framework for collaboration: an agreement on information exchange in terms of protocol, syntax and semantics, distribution of roles, responsibilities and labour, and determining how to coordinate, monitor and mitigate the progress of collaborative efforts. Past experience has taught that it is hard to enforce an á priori set of standards and systems, as there will always be stakeholders that have made different choices that better suit their mission. Addressing this concern by an open paradigm, that is capable of operating a mixed model organization makes a very desirable proposition and, moreover, has been proven feasible in a number of cases.

Surveying the propositions that have been discussed in the previous chapters, the thought whether it would be feasible to compose a system of systems where each of the surveyed candidates could find a meaningful role comes to mind. As noted in section 5.3, each candidate system has its individual strengths and weaknesses, the strengths of which could be leveraged in a combined system. Figure 37 presents a tentative combined WFGM system of systems.

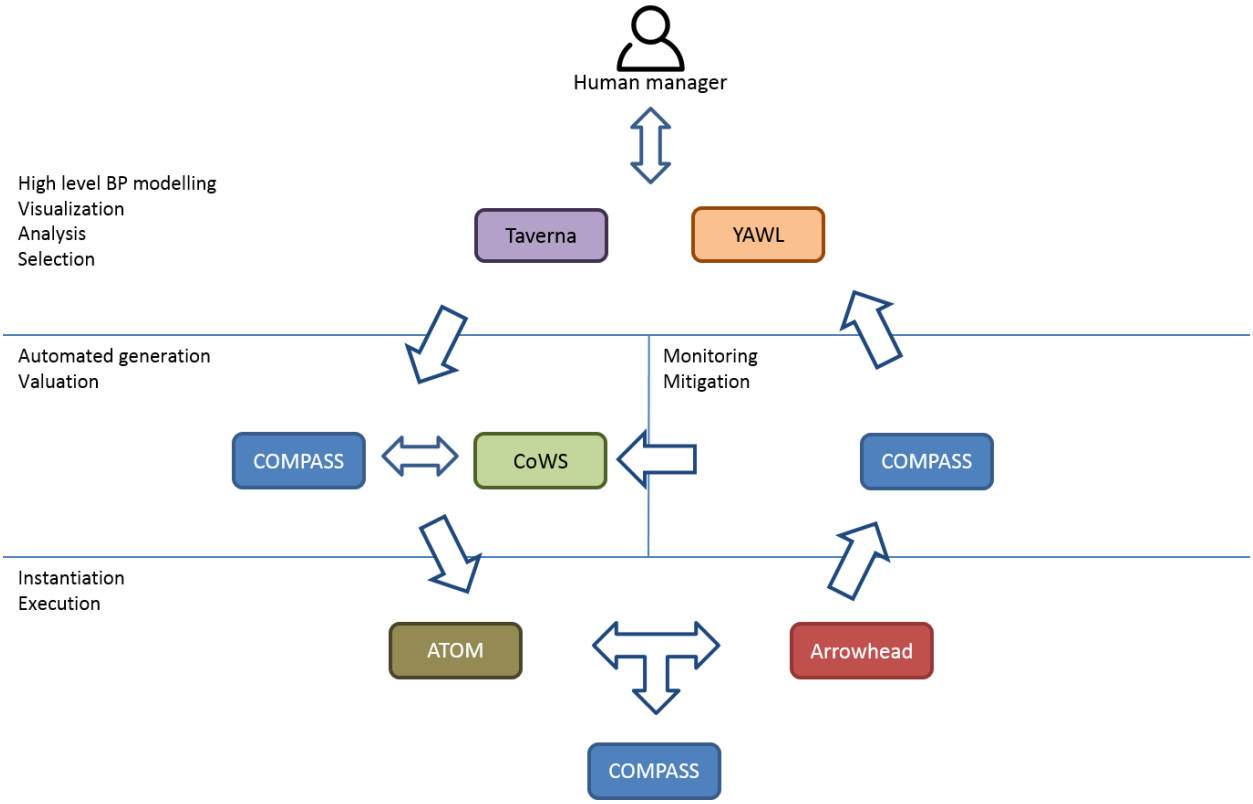


Figure 37: Combined WFGM System of Systems

At this point in time, it is impossible to estimate whether that system is feasible and how much effort it will take to implement. Nevertheless, validating at least part of it would be a useful exercise in Productive4.0.

7.3 Conclusions

In Figure 37 COMPASS is present in three of the four domains. In previous efforts, ATOM, COMPASS and CoWS have been configured to collaborate using BRAWL. Starting with this baseline the effort of realizing a collaborative WFGM system of systems, satisfying the requirements from chapter 3, the integration of systems such as Arrowhead, Taverna and YAWL and deploying the integrated systems in the Digital Product Footprint use case becomes a desirable target for Productive4.0.

8. References

1	Taylor, Frederick W., “Shop Management”, 1911, Harper and Bros., New York.
2	Gantt, Henry L., Organizing for Work, 1919, Harcourt, Brace, and Howe, New York. Reprinted in 1973 by Hive Publishing Company, Easton, Maryland.
3	http://en.wikipedia.org/wiki/Workflow
3.1	A. H. M. ter Hofstede (Editor), W. van der Aalst (Editor), M. Adams (Editor), N. Russell (Editor), “Modern Business Process Automation: YAWL and its Support Environment”, November 30, 2009
4	W. van der Aalst, K. van Hee, “Workflow Management: Models, Methods and Systems”, MIT Press, 2004, ISBN 026201889
5	The BPM tool ARIS Express can be obtained for free from: http://www.ariscommunity.com/aris-express
6	IDS Scheer AG website: http://www.ids-scheer.com/index.html
7	M. Stumptner, “An overview of knowledge-based configuration”, AI Communications, 10(2):111–125, 1997.
8	A. Günter and C. Kühn. “Knowledge-based configuration- survey and future directions”. XPS-99: Knowledge-Based Systems, pages 47–66, 1999.
9	L. Anselma and D. Magro. “Dynamic problem decomposition in configuration”. In Proceedings of the Configuration workshop held at IJCAI 2003, Acapulco, Mexico, pages 21–26, 2003.
10	P. Albert, L. Henocque, and M. Kleiner. “Configuration-based workflow composition”. In: Proceedings of the IEEE International Conference on Web Services (ICWS ’05), pages 285–292, Washington, DC, USA, 2005. IEEE Computer Society.
11	M. I. Campbell, J. Cagan, and K. Kotovsky.“A-design: An agent-based approach to conceptual design in a dynamic environment”. Springer London, 11(3):172–192, October 1999.
12	G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van der Velde, and B. Wielinga. “Knowledge engineering and management: the CommonKADS methodology”. MIT Press, Cambridge, Mass., 2000.
13	P. H. G. van Langen. “The Anatomy of Design: Foundations, Models and Applications”. PhD thesis, Vrije Universiteit Amsterdam, 2002.
14	C. Wargitsch, T. Wewers, and F. Theisinger. “Workbrain: Merging organizational memory and workflow management systems”. In: Workshop of Knowledge-Based Systems for Knowledge Management in Enterprises at the 21st annual German Conference on AI (KI-97, pages 214–219. DFKI, 1997.
15	J. Peer. “Web service composition as AI planning - a survey”. Technical report, University of St. Gallen, 2005.
16	J. Rao and X. Su. “Semantic Web Services and Web Process Composition”, volume 3387 of LNCS, chapter A Survey of Automated Web Service Composition Methods, pages 43 – 54. Springer-Verlag Berlin, 2005.
17	F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. “Adaptive and dynamic service composition in eFlow”. In: B. Wangler and L. Bergman, editors, Advanced Information Systems Engineering: 12th International Conference, CAiSE 2000, Stockholm, Sweden, June 2000. Proceedings, volume 1789 of LNCS, pages 13–31, 2000.
18	K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma. “Framework for semantic web process composition”. International Journal of Electronic Commerce, 9(2):71–106, 2005.
19	A. J. S. Cardoso. “Quality of Service and Semantic Composition of Workflows”. PhD thesis, The University of Georgia, August 2002.
20	S. van Splunter. “Automated Web Service Reconfiguration”. PhD thesis, VU University Amsterdam, March 2010.

21	E. Sirin, B. Parsia, and J. Hendler. “Template-based composition of semantic web services”. In AAAI Fall Symposium on Agents and the Semantic Web, Virginia, USA, November 2005.
22	N. Lin, U. Kuter, and E. Sirin. “Web service composition with user preferences”. The Semantic Web: Research and Applications, pages 629–643, 2008.
23	A. Bartoli, R. Jiménez-Peris, B. Kemme, C. Pautasso, S. Patarin, S. Wheater, and S. J. Woodman. “The ADAPT framework for adaptable and composable web services”. IEEE Distributed Systems online, Web Systems section, September 2005.
24	D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. “Managing escalation of collaboration processes in crisis mitigation situations”. In Proceedings of the International Conference on Data Engineering, page 45, Los Alamitos, CA, USA, 2000. IEEE Computer Society.
25	D. Karastoyanova and A. Buchmann. “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, pages 27–29, Jakarta, Indonesia, September 2004.
26	C. Larman. “Applying UML and Patterns”. Prentice Hall, 2nd edition, 2002.
27	OMG. “Unified Modeling Language: Superstructure”, OMG Document nr. formal/2011-08-06, 2011
28	N. J. Nilsson. “Principles of Artificial Intelligence”. Springer Verlag, 1982.
29	Y.M.I. Dirickx, S.M. Baas and B. Dorhout, “Operationele Research” (in Dutch), Academic Service, 1987.
30	Tsang, E.P.K.,“ Foundations of Constraint Satisfaction”, Academic Press, London and San Diego, 1993, ISBN 0-12-701610-4
31	J. B. van Veelen. “SMDS: A top-down approach to self-management for dynamic collaboration systems”. In: SEAMS ’06: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, pages 58–64, New York, NY, USA, 2006. ACM Press.
32	FIPA. “Fipa agent management specification”. Technical report SC00023K, FIPA, 2004.
33	C. van Aart. “Organizational Principles for Multi-Agent Architectures”. Whitestein Series in Software Agent Technologies. Birkhäuser, Basel, Switserland, 2005.
34	http://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=325&ct=2
35	D.S. Alberts, J.J. Garstka and F.P. Stein, “Network Centric Warfare”, 1999, DoD CCRP, http://www.dodccrp.org/files/Alberts_NCW.pdf
36	United States Airforce Scientific Advisory Board, “*Report on Building the Joint Battlespace Infosphere*”, Vol 2, “*Interactive Information Technologies*”, 1999, SAB-TR-99-02, http://archive.adaic.com/ASE/ase02_01/bookcase/sprpts/jbi/files/JBI_Volume_2_Interactive_Information_Technologies.pdf
37	V.T. Combs cs., “*Joint Battlespace Infosphere: Information Management within a C2 Enterprise*”, 2005, http://www.dtic.mil/dtic/tr/fulltext/u2/a463694.pdf
38	http://ushahidi.com/
39	Heather Blanchard, Andy Carvin, Melissa Elliott Whitaker, Merni Fitzgerald, Wendy Harman, Brian Humphrey, Patrick Philippe Meier, and Catharine Starbird, “*The Case for Integrating Crisis Response with Social Media*”, American Red Cross White Paper. See: http://www.scribd.com/doc/35737608/White-Paper-The-Case-for-Integrating-Crisis-Response-With-Social-Media.
40	R. Brooks, "*A robust layered control system for a mobile robot*", 1986, IEEE Journal of Robotics and Automation, Vol. 2, Issue 1, pp. 14–23
41	S. E. Conry, D. J. MacIntosh, and R. A. Meyer. 1990. DARES: a distributed automated reasoning system. In *Proceedings of the eighth National conference on Artificial*

	intelligence - Volume 1 (AAAI'90), Vol. 1. AAAI Press 78-85.
42	L. Serafini and A. Tamilin. DRAGO: Distributed Reasoning Architecture for the Semantic Web. In Proc. of the Second European Semantic Web Conference (ESWC'05), Springer-Verlag, 2005.
43	Smith, R. G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Trans. on Computers C-29(12):1104-1113 (1980)
44	Parker, L.E., "*ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation*", IEEE Trans. on Robotics and Automation, 1998, 14(2):220-240
45	Common Object Request Broker Architecture (CORBA), see http://www.corba.org/ and http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
46	http://en.wikipedia.org/wiki/Java_(programming_language)
47	http://www.cougaar.org/
48	Java Agent Development Framework JADE, see http://jade.tilab.com/
49	http://www.agentscape.org/
50	http://eve.almende.com/
51	David G. A. Mobach, Benno J. Overeinder and Frances M. T. Brazier, A Resource Negotiation Infrastructure for Self-Managing Applications, in: Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC 2005), pages 381--382, IEEE, 2005
52	M. A. Oey, R. J. Timmer, D. G. A. Mobach, B. J. Overeinder, and F. M. T. Brazier. 2007. WS-agreement based resource negotiation in AgentScape. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems* (AAMAS '07). ACM, New York, NY, USA, DOI=10.1145/1329125.1329448
53	Koenig, S., Keskinocak, P., and Tovey, C., 2010, Progress on agent coordination with cooperative auctions, in Proceedings of AAAI Conference on Artificial Intelligence, pp. 1713-1717
54	E. Bonabeau, M. Dorigo, G. Theraulaz, 1999, "*Swarm Intelligence; from natural to Artificial Systems*", Santa Fe Institute, ISBN 0-19-513159-2
55	G. Bieber, J. Carpenter, "OpenWings a service-oriented component architecture for self-forming, self-healing network centric systems"
56	http://en.wikipedia.org/wiki/Service-oriented_architecture
57	T. Erl, 2005, "Service-Oriented Architecture; Concepts Technology and Design", Prentice Hall, ISBN 0-13-185858-0
58	http://en.wikipedia.org/wiki/Internet_of_Things
59	http://en.wikipedia.org/wiki/Cloud_computing
60	http://en.wikipedia.org/wiki/Software_as_a_Service
61	http://en.wikipedia.org/wiki/Smart_grid
62	http://en.wikipedia.org/wiki/Smart_city
63	http://en.wikipedia.org/wiki/Industry_4.0
64	Boyd, 1995, "*The essence of winning and loosing*", 1995, http://www.danford.net/boyd/essence.htm, see also: http://en.wikipedia.org/wiki/OODA_loop
65	Kephart (?), 2005, IBM Autonomic computing, http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
66	J.O. Kephart, D.M. Chess, 2003, "*The vision of autonomic computing*", IEEE Computer, Vol. 36, pp41-50
67	H. Mintzberg, "*Structures in Fives; Designing effective organizations*", 1983, Prentice Hall, ISBN 0-13-854191-4
68	"Coping through a Disaster: Lessons from Hurricane Katrina," Journal of Homeland Security and Emergency Management: Vol. 8: Iss. 1, Article 19. DOI:10.2202/1547-7355.1836
69	GRAAP Working Group. WS-Agreement Specification. Technical report, Open Grid Forum, October 2006. Available at: http://www.ogf.

	org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal_sp_tn_jpver_v2.pdf
70	A. Keller, H. Ludwig, 2003, The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services/ in *J. of Network and Systems Management*, Plenum, Vol. 11, No 1, 2003, pp. 57-81.
71	Oliver Wäldrich, Dominic Battré, Frances M. T. Brazier, Kassidy P. Clark, Michel A. Oey, Alexander Papaspyrou, Philipp Wieder and Wolfgang Ziegler, WS-Agreement Negotiation: Version 1.0, Open Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, 2011
72	F. Heidari, P. Loucopoulos, Quality evaluation framework (QEF): Modeling and evaluating quality of business processes, International Journal of Accounting Information Systems, Available online 14 October 2013, ISSN 1467-0895, http://dx.doi.org/10.1016/j.accinf.2013.09.002
73	http://www.prismtech.com/vortex/vortex-opensplice
74	https://hazelcast.com/
75	https://redis.io/
76	http://www.w3.org/TR/wsdl20/
77	Leymann, Frank. (2001). "Web Services Flow Language (WSFL 1.0)". IBM Corporation
78	http://www.w3.org/TR/xpath/
79	http://www.w3.org/TR/xslt/

9. Appendix

9.1 Abbreviations

Table 2: Abbreviations

Abbreviation	Meaning
BRAWL	BRIDGE Annotated Workflow Language (Result of FP7 Project BRIDGE)
CWFGM	Collaborative Workflow Generation and Management
DPF	Digital Product Footprint (Productive4.0 use-case)
IIoT	Industrial Internet of Things
IoT	Internet of Things
LSP	Logistics Service Provider
PLM	Product Lifecycle Management
ePLM	extended Product Lifecycle Management
QoS	Quality of Service
QoSM	Quality of Service Management
SCM	Supply Chain Management
SLA	Service Level Agreement
SoS	System of Systems
WF	Workflow
WFGM	Workflow Generation and Management

9.2 BRAWL Syntax

9.2.1 BRAWL basis syntax

The BRidge Annotated Workflow Language (BRAWL)

(DRAFT)

Authors: J.B. van Veelen, Thales Research & Technology Netherlands
N.J.E. Wijngaards, Thales Research & Technology Netherlands
S. van Splunter, Delft Technical University
A. Stam, Almende B.V.

Abstract

This document specifies the BRIDGE Annotated Workflow Language, which is used by the Workflow Generation mechanisms, Workflow instantiation mechanisms and the Workflow Quality of Services Management mechanisms in BRIDGE-based Agile Response Systems.

Goal of BRAWL

In complex dynamic organizations situated in a dynamic environment we are faced with the challenge of configuration and coordination. The questions are (1) how to coordinate the activities of multiple agencies, while maintaining the integrity and personal policies of each individual participant, and (2) how to configure each actor and agent to contribute to the collaboration effectively and efficiently. In BRIDGE Agile Response systems (BARS), which are a class of complex dynamic organizations, we aim to provide automated coordination and configuration mechanisms, in the form of smart workflow management systems. The workflow management systems differ for each agency and potentially for each goal and collaborate to achieve suitable coordination of activities.

The BRIDGE Annotated Workflow Language (BRAWL) is an agreement on the format of information exchange between the workflow management systems. BRAWL defines syntax and semantics of workflow elements. BRAWL allows the specification of incomplete workflows, which can be further completed by any of the available workflow management systems. The BRAWL language is positioned as a platform for communication and collaborative workflow management.

Conventions

The BRAWL derives its style and syntax conventions from the OMG's XML specification. In this document we will use the EBNF notation to specify the syntax elements of BRAWL. The BRAWL will be defined top-down that is, starting from the Bridge Annotated Workflow down to the smallest elements.

Notation conventions:

Expressions:

`normal-expression`
: non-terminal, previously introduced in the syntax specification

`"literal-expression"`
: exact wording or notation to be used in a BRAWL expression

`bold-expression`
: Non-terminal, defined further on in the syntax specification

Specification:

`x ::= y` nonterminal `x` is specified as expression `y`

Constructors

`x y`	sub-expression `y` follows sub-expression `x`
`[x]`	sub-expression `x` may occur zero (0) or one (1) time
`{x}`	sub-expression `x` may occur zero (0) or more times
`x | y`	either sub-expression `x` or sub-expression `y` occurs
`(x y)`	groups sub-expressions `x` and `y` into a single sub-expression for the '`[…]`', '`{…}`' and '`…|…`' constructors

The core of BRAWL is designed to be simple and contains just a few tokens. However, BRAWL is also set up to be modular, implying that new modules containing new tokens for special concerns can be included. So, the basic version of BRAWL includes all the keywords and constructs required to specify annotated workflows, scripts and references to other workflows and modules. The BRAWL modules are intended to contain the tokens and keywords for specific issues, for example, trust, policies or security.

Annotations may include tokens that a certain workflow interpreter cannot understand. The principle is that the statements containing these tokens are ignored and, importantly, left unmodified by the interpreter. These statements will be processed by other management processes that can understand these tokens. The principle of 'leave the stuff you don't understand alone' provides us with the ability to enforce a clean separation of concerns between the management processes.

BRAWL Specification

This chapter specifies the BRIDGE Annotated Workflow and its components: the Banner, the Preamble and the Workflow.

BRIDGE Annotated Workflow

The BRIDGE Annotated Workflow is the expression we aim to specify using BRAWL. We want to be able to use the BRIDGE Annotated Workflow during the generation process, the instantiation process and the quality management process. Other usages of the BRAWL Annotated Workflow may be visualization in composition, selection or decision making processes involving human operators.

We define the BRIDGE Annotated Workflow to be an expression composed of a Banner, a Preamble and a Workflow. All three of these components occur not more than once in the expression and are compulsory.

The banner identifies the expression to be a BRAWL annotated workflow. The preamble contains information concerning the workflow specified in the workflow part. The BRAWL workflow part contains the specification of the actual workflow.

The EBNF definition of the BRAWL Annotated Workflow:

```
BRIDGE Annotated Workflow ::=
          Banner
          "<brawl" [brawl_id] ">"
           Preamble
           Workflow
          "</brawl>"
Banner ::=
          "<!doctype brawl version" major.minor{.subminor}">"

          major ::= numeric
          minor ::= numeric
          subminor ::= numeric

brawl_id ::=
           "brawl_" identifier

Comment ::=
          "<comment " {printable} ">"
          Comments can occur anywhere in a BRAWL expression, except within another (key-) word.
          Comments are ignored by the parser.
```

Preamble

The Preamble of a BRAWL Annotated Workflow documents the aspects that are global to the workflow. These aspects include the unresolved targets (in case the workflow is a partial workflow), the(current) value of a (partial) workflow (in terms of cost, number of resources deployed, precision, predicted load, throughput or other terms) and other attributes encoded in (global) annotations.

The 'targets' are essential for the workflow generation and workflow modification processes, as these targets describe the changes, replacements, modification, refinements, extensions etcetera on the workflow provided in the rest of the BRAWL document. Note that, when **no** targets are present, apparently there is no more work to be done on constructing the workflow, and thus the workflow can be executed or used to fill a target in a partial workflow.

```
Preamble ::=
          "<preamble>"
            {module-inclusion}
            {cost}
            [target-list]
            {global-quality}
            {optional-preamble-element}
          "</preamble>"

module-inclusion ::=
          "<include>"
           module-name
           {"," module-name}
          "</include>"

module-name ::=
          identifier

cost ::=
```

```
            "<cost> "
              name-value
            "</cost>"

target-list ::=
            "<targets>"
              target
              {target}
            "</targets>"
target ::=
            "<target>"
              identifier " = "
                ( ("action " action-descriptor)
                | ("artefact " artefact-descriptor)
                )
              {constraint}
            "</target>"
constraint ::=
            "<constraint>"
              name-value
            "</constraint>"
```

Instead of using the descriptors (There may be many identical descriptors in a workflow, hence this target-list refers to workflow-elements ambiguously) we propose to use the identifiers.
In that case we have to generate the action and artefact identifiers at the point where we find out we need them, not at the time where we find out how to implement them.

```
global-quality
            "<quality>"
              name-value
            "</quality>"
```

The non-terminal *action-descriptor* is described in the sub-section "Action" further on in this document. The non-terminals *artefact-descriptor* and *name-value* is documented in the sub-section "Artefact" further on in this document.

Optional Preamble elements

The optional preamble elements include references to the owner and the generator of the workflow. Also a reference to a compositional workflow that encapsulates this workflow can be included.

```
optional-preamble-element::=
              owner_reference
            | generator_reference
            | sub_workflow_reference

owner_reference ::=
            "<owner_reference>"
              string
            "</owner_reference>"

generator_reference ::=
            "<generator>"
              string
            "</generator>"

sub_workflow_reference ::=
              sub_workflow_declaration
            | realization_of_declaration

sub_workflow_declaration ::=
```

```
            "<is_realization_of>"
              brawl_id
              swf_decl_id
            "<is_realization_of>"

swf_decl_id ::=
            identifier
```

BRAWL facilitates compositionality in the workflow specification. What we want is that we can specify an action in a workflow, but the action actually 'hides' a sub-workflow.
The way to achieve this is by including a reference in the preamble of the sub-workflow to the encompassing workflow like:
`<is_realization_of> brawl_id swf_decl_id </is_realization_of>`
and include a reference to the brawl_id of the sub-workflow in the action that hides it, by a statement:
`<composition swf_decl_id realized_by brawl_id>`
Since a large workflow may contain multiple composed sub-workflows, we need a reference to the location in the workflow that the sub-workflow specifies. The location in the workflow is uniquely identified by `swf_decl_id`.

Workflow

The Workflow is specified as a list of Actions, Artefacts, Arrows, Resources and Annotations.

```
Workflow ::=
            "<workflow" [workflow-id] ">"
              {  action
               | artefact
               | arrow
               | resource
               | annotation
              }
            "</workflow>"

workflow-id ::=
            "workflow_" identifier
```

Next we will define the components of annotated workflows: Actions, Artefacts, Arrows, Resources and Annotations.

Action

The action sub-expression is used to describe *operations*, *actions* or *processes*. Operations, actions or processes produce artefacts, and may use artefacts to process as inputs.

A BRAWL Action contains at least an identifier and an action-descriptor; it may contain many optional action elements.

```
action ::=
            "<action" action-id ">"
             descriptor
             {optional action element}
            "</action>"

action-id ::=
            "action_" identifier

descriptor ::=
            "<descriptor>"
             identifier
            "</descriptor>"
```

Optional action element

The optional action elements include a reference to the generating mechanism, a composition declaration and an action extension. An action extension is to be defined in a (separate) module; this module has to be included in the preamble of the BRWAL expression, or parsing the expression will fail

A composition declaration acts as a placeholder for a sub-workflow that is defined in another BRAWL expression.

```
Optional action element ::=
            generator_reference
          | composition_declaration
          | action_extension
```

Action-extensions are defined in BRAWL Modules.

```
generator_reference ::=
          "<generated_by>"
            string
          "</generated_by>"
```

We have added the reference to the generator of this particular element of the workflow; that will also be used for artefacts, annotations and arrows. Like mentioned before, it is as yet unclear how to formulate a reference to a process; we could just use something like an IOR (from CORBA), or something that contains more semantics like a structure:

```
<process>
  <name string>
  <BRIDGE_id identifier>
  <host hostname>
  <port portnumber>
</process>
```

So this is up for discussion

```
composition_decaration ::=
          "<composition"
            swf_decl_id
            ["realized_by"
            brawl_id]
          ">"
```

This construct is the counterpart of the sub-workflow reference in the preamble. That is, a sub-workflow reference in a preamble must correspond to a composition declaration in the workflow that is referred to.

Recursive or circular references are not allowed.

In general, there will be many more aspects that need to be declared regarding an action, for example a requirement or a quality of service. In BRAWL these additional aspects are documented in Annotations.

Artefact

Artefacts are produced by actions and are used to serve as inputs for other actions, possibly actions in other workflows (this requires the buffering of the artefact(s) in repositories).

A BRAWL Artefact contains at least an identifier and an artefact-descriptor; it may contain many optional artefact elements.

```
artefact ::=
          "<artefact" [artefact-id] ">"
           descriptor
           {optional-artefact-element}
          "</artefact>"

artefact-id ::=
```

```
            "artefact_" identifier
```

Optional artefact element

Like actions, artefacts can have a reference to a generator and include possible extensions, defined in modules. See section "Modules".

```
optional-artefact-element ::=
              generator_reference
            | artefact_extension

generator_reference ::=
            "<generated_by>"
              string
            "</generated_by>"
```

Arrow

An arrow describes the flow of information (in the form of artefacts) from and to actions.

```
arrow ::=
            "<arrow" [arrow-id] ">"
                ("<from " action-id "><to " artefact-id ">")
              |("<from " artefact-id "><to " action-id ">")
            "</arrow>"

arrow-id ::=
            "arrow_" identifier
```

The semantics of an arrow can be loosely defined as "*used by*" (in case of an artefact-to-action arrow) or "*produces*" (in case of an action-to-artefact arrow). The arrow contains no other aspects that the elements it connects, hence relevant other aspects of arrows have to be describe using annotations.

An Arrow does not have a generator reference or possible extensions. However, many Annotations may be associated to an Arrow.

Resource

To execute actions, we need resources, both to perform the execution and to supply the execution with required 'ingredients'. The resources come in two types, active resources (like human actors, software agents and equipment) and passive resources (expendable resources like bandwidth, time money and supplies).

Active resources can execute actions, own (possess, keep) artefacts and have annotations. Passive resources may have annotations. Passive resources are expressed as budgets.

Active Resources

Active resources are divided into three groups: actors (human operators) agents (software beings) and services (service includes devices and tools). Actors and agents may use services to accomplish an action. Actors, agents and services may use a budget (an amount of some passive resource) during the execution of an action.

```
resource ::=
              active_resource
            | passive_resource

active_resource ::=
              actor
            | agent
            | service

actor ::=
            "<actor" [active_resource-id] ">"
```

```
            descriptor
            active_resource_reference
            { optional_active_resource_element
            | optional_actor_element}
          "</actor>"
agent ::=
          "<agent" [active_resource-id] ">"
            active_resource_descriptor
            active_resource_reference
            { optional_active_resource_element
            | optional_agent_element}
            "</agent>"
service ::=
          "<service" [active_resource-id] ">"
            active_resource_descriptor
            active_resource_reference
            { optional_active_resource_element
            | optional_service_element}
          "</service>"
active_resource-id ::=
          identifier
```

Optional active resource elements

For doing proper sub-system management, we require information on a few vital resource attributes. We could have left these to annotations, but instead, they are included as optional parts of an active resource definition. These attributes are the resource's status, capabilities and its current schedule. Further an active resource may be extended with new attributes, but we advise to consider carefully whether these are not better documented as annotations.

```
optional_active_resource_element ::=
            status
          | capability
          | schedule
          | resource_extension

status ::=
          "<status>"
            status_value
          "</status>"

status_value ::=
            "available"
          | "off-line"
          | "error"
          | "needs_input"
          | "needs_resource"
          | "waiting"
          | "busy"

capability ::=
          "<capability>"
            capability_descriptor
          "</capability>"

capability_descriptor ::=
          action-descriptor
          capability_attribute {capability_attribute}

capability_attribute ::=
            skill_attribute
          | precision_attribute
          | performance_attribute
          | quality_attribute
          | other_attribute
```

```
schedule ::=
           "<schedule>"
             schedule-entry
            {schedule-entry}
           "</schedule>"

schedule-entry ::=
           "<entry>"
            entry-indicator
            action-id
           "</entry>"

entry-indicator ::=
           startcondition [stopcondition]

startcondition ::=
           condition

stopcondition ::=
           condition
```

Passive Resource

Passive resources are available budgets

```
passive_resource ::=
           "<budget" [budget-id] ">"
            descriptor
            budget_amount
            budget_unit

            {optional budget element}
           "</budget>"



budget_amount ::=
           "<amount>"
            number
           "</amount>"

budget_unit ::=
           "<unit>"
            identifier
           "</unit>"
```

Annotation

The annotation is a very powerful, yet simple construct to associate values to the other workflow elements. The basic structure of the annotation is nothing else than an identifier unique to the workflow, and a name-value pair. While the name is constrained to be a single word, the value can be anything that can be expressed.

```
annotation ::=
           "<annotation" annotation-id ">"
             "<belongs-to>"
                 ("action" action-id)
               | ("artefact" artefact-id)
               | ("arrow" arrow-id)
             "</belongs-to>"
             name-value
           "</annotation>"

name-value ::=
           "<name>" keyword "</name>"
           "<value>"
            (
               [delimiter] anything [delimiter]
```

```
              | "link:" reference
              | script
              | composed_value
             )
           "</value>"

annotation-id ::=
           "annotation_" identifier

global-annotation-id ::=
           "global_" identifier

script ::=
           "<script>"
            s-expression
           "</script>"

composed_value ::=
           <composition>
            name-value {name-value}
           </composition>

keyword ::=
           printable {printable}

reference ::=
           string
```

The name of the name-value pair can be chosen freely by the systems' designer and has a semantics that is defined in the context of a particular systems' design. The section Annotation keys describes all of reserved keywords that are meaningful in the context of the processes of generation, manipulation and instantiation of workflows or the quality management of instantiated workflows. All other keywords are keyword that may have meaning in a user defined context and will be ignored by the above mentioned mechanisms.

The value part of an annotation may contain a numerical or scalar value (like "9" or "red") or any desired structured, arbitrary complex value (such as:

```
"{Stanley Clarke,
  East River Drive,
  {Track-list
   {1: Justice's Groove From The Columbia Motion Picture "Poetic Justice"},
   {2: Fantasy Love},
    …
   {11: "Lords Of The Low Frequencies"},
   {12: Funk Is Its Own Reward}
  },
  Audio CD,
  {August 24, 1993},
  {Number of Discs: 1},
  {Label: Sony},
  {ASIN: B0000027LH}
 }"
```

If required one could include a Shakespeare play, a service contract or a database.). To include non-structured elements (such as audio, video or bit-sequences) use the *link*-construct.

Mathematic expressions

In BRAWL we include just a few essential operators to be used in requirements, constraints and maybe other expressions. The aim of the symbolic expressions in BRAWL is to be able to formulate indirect values, i.e. values which need to be computed (at run-time) and possibly depend on the current value of variables. The values should be easy to compute, so just some very basic mathematical, logical and comparison operators are offered.

Constructs

To construct expressions we first introduce the concepts of sequencing operators, terms, and lists.

List

syntax

```
list ::=
 "{"
     s-expression
     {"," s-expression}
 "}"
```

description

List of stuff.

- Need to include reference to list as valid list.
- Introduce List-variable @x?

Term

syntax

```
term ::=
  variable
| value
| operation
| "(" term ")"


variable ::=
  "$" name
| dotted_reference

value ::=
  numeric
| boolean
| string
| time

dotted_reference ::=
 "["
    ( "action" action_id)
  | ("artefact" artefact_id)
  | ("arrow" arrow_id)
 "]" {"." keyword}

numeric ::=
  "0" ... "9" {"0" ... "9"}

time ::=
  "T" { "0" ... "9" }
```

description

A term is the basic form of an operand. An operand is an argument to an operation or an s-expression.

A variable takes the form of '`$x`'. We can add another variable variant for lists, for example '`@x`'; in that case the form '`$x`' becomes the form for scalar variables.
The scope of variable is always the entity the annotation is associated to; this means we can declare/initialize a variable in one annotation and use it in another.

- As a discussion point: do we want to include a third truth value "UNKNOWN"? This might come in handy during the WF generation process, but what are the implications of encountering an "UNKNOWN" in a logical evaluation or a conditional statement?

Operation

syntax

```
operation::=
  arithmetic-operation
| logic-operation
| selection
| comparison
| "(" term ")"
```

description

None.

Sequence

syntax

```
sequence::=
 "begin"
   {s-expression ";"}
 "end"
```

description None.

S-expression

syntax

```
s-expression::=
  assign
| forall
| condition
| sequence
| return
| empty
```

description None.

The basic s-expressions cater for assignment, list-iteration and conditional execution. Furthermore there are the special functions return, and empty.

Assign

syntax

```
assign ::=
  variable ":=" s-expression
```

description Evaluates s-expression and assigns the result to variable.
Variable has been defined with Term.

Forall

syntax

```
forall ::=
"forall " variable
   " in " s-expression (* 1 *)
   " do " s-expression (* 2 *)
   "end"
```

description Applies the third operand (s-expression 2) to occurrences of the first operand (x) in the second operand (s-expression 1).

- Is it useful to have a list instead of s-expression 1; this means that the forall statement would return a list
- Discussion between Sander and Bernard has led to the conclusion that operating on a list (replace s-expression 1) by a list has a number of advantages during both the generation and execution phase. For example, pushing the start-time by 1 minute (due to a 1 minute delay) for all flow-dependent nodes can be realizing by having a call that lists all the flow depending actions (at-runtime) so you don't have to enumerate them at generation time. Which makes you don't need to know everything at this point in generation (nice for COMPASS/SMDS and CoWS) and can postpone the generation of realizations (sub-workflows for composite actions) (nice for Almende)

Condition

syntax

```
condition ::=
"if " term
  " then " s-expression (* 1 *)
  " else " s-expression (* 2 *)
  " end"
```

alternative form:

```
condition::=
 "condition"
    {term ":" s-expression ";"}*
 "end"
```

description This expression will evaluate s-expression 2 if term evaluates to FALSE or 0. It will evaluate s-expression 1 in all other cases.

Alternative form: the condition will evaluate the list of terms, starting at the top, **until** it finds a term that is not FALSE or 0; it evaluates the s-expression associated to this term, **and will ignore the rest of the list of conditional clauses**. Hence if the list contains multiple terms that evaluate to TRUE or a value not 0, only the s-expression of the top-most one will be evaluated.

- Maybe this is too difficult for our BRAWL??

Return

syntax
```
return ::=
"return " term ";"
```

description The return statement makes that the value resulting in the evaluation of term is returned as the result-value of the current script context.
Hence, the s-expressions that come after the return statement are parsed (!!) but ignored.

Empty

syntax
```
empty ::=
  whitespace
```

description None.

Arithmetic operations

The arithmetic operations in BRAWL can perform some very basic mathematics. The limited set of operators is intended to satisfy the typical needs for mathematics in workflow annotations.

Arithmetic

syntax
```
arithmetic-operation::=
  term infix-arithmetic-operator term
| prefix-arithmetic-operator term
```

description Simple arithmetic operators to do some calculations, currently just +, -, * and /

Arithmetic operators

There are no shocking elements in this set of operators, use is as expected, may have to adapt the description somewhat…

Plus

token +
position infix, prefix
description Result is the addition of the value of its operands.
Ignored if used in prefix form.

Minus

token -
position prefix, infix
description Subtracts (the value of) right-hand operand from (the value of) the left-hand operand.
If used prefix, negates the numerical value of its operand.

Times

token *
position infix
description multiplies its operands

Divide

token	`/`
position	infix
description	divides (the value of) the left-hand operand by (the value of) the right-hand operand.

Logical operations

The logical operations allow us to evaluate expressions containing first order logic. Nothing complex.

The only thing slightly noteworthy is that BRAWL uses 0 and FALSE interchangeably (whatever suits best) and this implies that TRUE corresponds to (NOT 0).

Logic operations

syntax	`logic-operation::=`
	`  term infix-logic-operator term`
	`| prefix-logic-operator term`

description	Simple logic operators to do some calculations, currently just AND, OR, NOT

Logic operators and values

In this section we define the logic values and operations. We will start by the values TRUE and FALSE. In logic operations, FALSE is equivalent to 0 and *empty*, while TRUE is equivalent to NOT FALSE.

Boolean Value

`boolean ::=` `"TRUE" | "FALSE"`

`FALSE` is equivalent to `0`, and also equivalent to `empty` (the empty list). This means that in comparisons and conditions `FALSE`, `0` and `empty` yield the same result. In a numerical context, implicitly converting truth values to numbers, `FALSE` will be represented as 0.
`TRUE` is equivalent to `NOT FALSE`. So anything that is not equivalent to `FALSE` is per definition `TRUE`. In a numerical context `TRUE` will be represented as 1.

Next we define the logic operators AND, OR and NOT.

AND

token	`and`
position	infix
description	multiplies its operands

OR

token	`or`
position	infix
description	multiplies its operands

NOT

token	`not`
position	prefix
description	multiplies its operands

Note Do not confuse the *iterator* `forall`, which evaluates an S-expression for all elements in a list, with the logic *universal quantifier* 'for all' (usually notated as "∀"), which has no equivalent in BRAWL.

Comparison operations

The comparison operations allow comparing expressions and making decisions in conditional clauses (See: condition). All comparison operators yield a Boolean result: TRUE or FALSE.

Comparison operators

Less than

token	<
position	infix
description	TRUE if (value of) left-hand operand is less than (value of) right-hand operand, FALSE otherwise.

Less than or equal to

token	<=
position	infix
description	TRUE if (value of) left-hand operand is less than or equal to (value of) right-hand operand, FALSE otherwise.

Equal

token	==
position	infix
description	TRUE if (value of) left-hand operand is equal to (value of) right-hand operand, FALSE otherwise.

Greater than or equal to

token	>=
position	infix
description	TRUE if (value of) left-hand operand is greater than or equal to (value of) right-hand operand, FALSE otherwise.

Greater than

token	>
position	infix
description	TRUE if (value of) left-hand operand is greater than (value of) right-hand operand, FALSE otherwise.

Selection operations

The selection operations select an element from a list. So, the *value* of one of the operands of the selection operators has to be a list. There is one function here that does not do selection, but just counts the elements in the list, fur convenience purposes.

Selection operators

Least

syntax	least::= "least (" **list** ")"
description	selects the 'smallest' element in list if the elements of list can be ordered, FALSE other wise

Most

syntax	most::= "most (" **list** ")"
description	selects the largest element in list if the elements of list can be ordered, FALSE other wise

Select-n

syntax	select-n::= "select (" **list** "," **term** ")"
description	Selects the n^{th} element from the list, where n derived from the evaluation of term. If the list has no nth element (i.e. n equals 0 or n is larger than the size of list), the operator returns FALSE.

Size

syntax
```
size::=
 "size (" list ")"
```
description Counts the elements in the list.

Head

syntax
```
head ::=
 "head (" list ")"
```
description Return first element of the list or FALSE if the list is empty.

Tail

syntax
```
tail ::=
  "tail (" list ")"
```
description Return list except first element of the list or FALSE if the list is empty.

Alphabet, delimiters and interpunction

This section contains the definition of all basic elements of BRAWL, such as identifiers and strings.

String

```
string::=
 quote printable {printable} quote

printable::=
   char
 | decimal
 | whitespace
 | punctuation
 | misc_token

char        ::= "a"| .. | "z" | "A" | .. | "Z"
decimal     ::= "0"| .. | "9"
whitespace  ::= " " | "\t" | "\n" {" " | "\t" | "\n"}
punctuation ::=
    {"."| "," | "!" | "?" | ":" | ";" | "(" | ")" | "/" }
misc_token ::= {"-" | "+" | "*" | "_" }
```

Identifier

```
identifier ::=
   char { char | decimal | misc_token }
```

Delimiter

```
delimiter::=
 quote | "'" | "#" | "%"
```

Quote

```
quote ::= """
```

Modules

In this section we discuss how modules are structured. A module allows a programmer to define new inherent attributes of an action, an artefact or a resource. In all cases the syntax structure is roughly the same.

The mechanism that parses the module should add internal structures to action, artefact or resource (respectively) to represent the added definitions.

Module syntax

Modules are provided in separate files, and hence constitute separate documents. Therefore, a BRAWL module consists of a banner, and a body; the body contains a number of definitions.

```
brawl-module ::=
        "<!doctype brawl-module"
          module-name
          "version" major.minor{.subminor}">"

        "<brawl-module>"
```

```
        module-definition
        {module-definition}

        "</brawl-module>"

module-definition ::=
          action_extension_def
        | artefact_extension_def
        | resource_extension_def
```

The idea behind a module is that it contains extensions on the previous definitions of actions, artefacts and resources. Extensions take the form of the definition of a new key-word, to be used in the context of an action-, artefact- or resource specification respectively, and a type for the value to be associated with the new keyword.

```
action-extension-def ::=
        "<extends action" action-descriptor ">"
         action-extension
        {action-extension}
        ">"

action-extension ::=
        "<key" action-keyword "type" keytype ">"

action-keyword ::=
        keyword

artefact-extension-def ::=
        "<extends artefact" artefact-descriptor ">"
         artefact-extension
        {artefact-extension}
        ">"

artefact-extension ::=
        "<key" artefact-keyword "type" keytype ">"

artefact-keyword ::=
        keyword

resource-extension-def ::=
        "<extends resource" resource-descriptor ">"
         extension
        {extension}
        ">"

resource-extension ::=
        "<key" resource-keyword "type" keytype ">"

resource-keyword ::=
        keyword

keytype ::=
          "number"
        | "boolean"
        | "string
        | "list"
        | "name-value"
        | "structured"
        | empty
```

Module usage

The definitions in a module can be used in a BRAWL expression using the syntax of the action_extension, artefact_extension and resource_extension.

```
action_extension ::=
       "<" action-keyword value ">"
```

```
artefact_extension ::=
       "<" artefact-keyword value ">"

resource_extension ::=
       "<" resource-keyword value ">"
```

The keywords used in these expressions need to be defined previously in a module. A keyword must also be used in the context that it is defined for, e.g. a keyword defined in an action_extension_def can only be used in an action specification. Finally, the type of the value used must match the value defined in in the corresponding extension definition.

Examples

Some examples of the BRAWL language that clarify issues that may have been unclear so far. The form of these examples will be to answer the question "How do I say using BRAWL … ?".

Basics

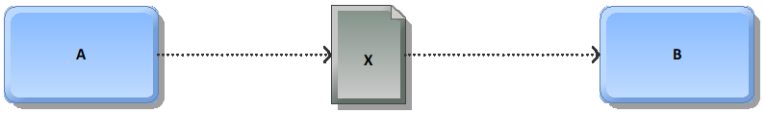
This first section demonstrates the basic use of BRAWL based on fairly simple examples. The first example illustrates how to specify a workflow composed of two actions and one artefact.

Example 1: Basic two-action workflow

What?

Action A produces Artefact X which is then used by Action B.

Schematic



Explanation

This first example shows a complete BRAWL expression. Note that the banner precedes everything and how everything else is neatly wrapped in proper tags. The typesetting of a BRAWL-file is generator specific.

The BRAWL specification starts with a comment, which will be ignored by the parser.

The preamble is rather empty. It just contains a reference to the generator of the BRAWL file.

The workflow defines the two Actions A and B (`action_A` and `action_B`) and the Artefact X (`artefact_X`). Next it defines the Arrows `arrow_1` and `arrow_2` that connect A to X and X to B respectively. Note that the identifiers of Actions, Artefacts, Arrows and Annotations can be freely chosen (although each has a specific prefix), but the identifiers must be unique within the context of the BRAWL expression.
This is required to be able to construct unambiguous cross-references in the expression. In this case the identifiers of the Actions and Artefact are used by the arrows.

BRAWL

```
<!doctype brawl>
<brawl brawl_001>

<comment This is a comment!>

<preamble>
  <generator>
    "Bernard v1.0"
  </generator>
</preamble>
```

```
<workflow workflow_1>

  <action action_A>
    <descriptor>
      "action A"
    </descriptor>
  </action>

  <action action_B>
    <descriptor>
      "action B"
    </descriptor>
  </action>

  <artefact artefact_X>
    <descriptor>
      "artefact X"
    </descriptor>
  </artefact>

  <arrow arrow_1>
    <from action_A>
    <to artefact_X>
  </arrow>

  <arrow arrow_2>
    <from artefact_X>
    <to action_B>
  </arrow>

</workflow>
</brawl>
```

The next example shows how to specify annotations to Actions, Arrows and Artefacts.

Example 2: Some Annotations

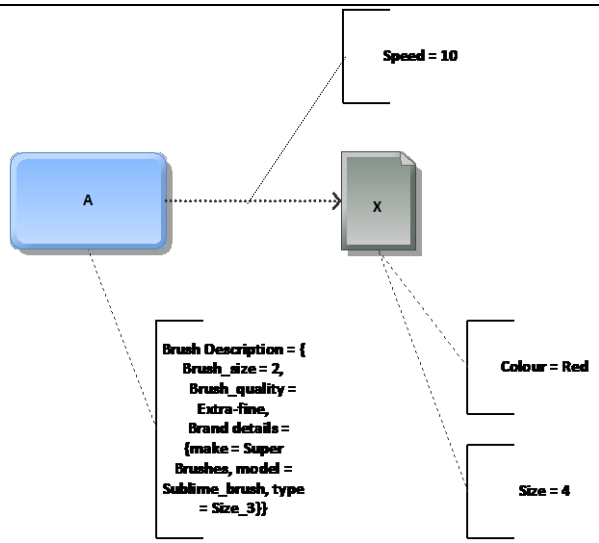
What?

In this example we present a very simple workflow of an Action A that produces an Artefact X.
Action A has a complex attribute (or property), which (presumably) describes the tool that is capable of executing the action. In this case it describes some characteristics of a brush.

Artefact X possesses to noteworthy properties, one that it has a Colour value Red and second that it has a Size of 4.

Finally, the Arrow has an attribute Speed with the value 10.

Schematic



Explanation

After the preliminary definitions in the preamble, Action A, Artefact X and the connecting Arrow are specified as in example 1.

The Arrow from A to X has a single annotation. The identifier of the Arrow is referred to in the belongs_to field of annotation 1. Note that not only the identifier but also the type (arrow) is included in the belongs_to field.

It is easy to attribute multiple properties to an element, as demonstrated by Artefact X, who possesses not one but two annotations, Annotation 2 and Annotation 3.

Action A has but a single, complex-valued Annotation, Annotation 4. Annotation 4 demonstrates how to use the composition –tags in the value part of the name-value to structure the value. Not that a composition is basically a list of name-values.
Also note that the placement of delimiters around values is optional. Typical you will want to use delimiters when the notation of the value contains whitespace or 'weird characters' that will cause the parser to terminate too early. Delimiters are quotes (' ' and " ") as well as the characters # and %.

BRAWL

```
<!doctype brawl>
<brawl brawl_002>

<preamble>
  <generator>
    "Bernard v1.0"
  </generator>
</preamble>

<workflow workflow_2>

  <action action_A>
    <descriptor>
      "action A"
    </descriptor>
  </action>

  <artefact artefact_X>
    <descriptor>
      "artefact X"
    </descriptor>
  </artefact>

  <arrow arrow_1>
    <from action_A>
    <to artefact_X>
  </arrow>

  <annotation annotation_1>
```

```
        <belongs_to arrow arrow_1>
        <name Speed>
        <value 10>
      </annotation>

      <annotation annotation_2>
        <belongs_to artefact artefact_X>
        <name Colour>
        <value 'Red'>
      </annotation>

      <annotation annotation_3>
        <belongs_to artefact artefact_X>
        <name Size>
        <value 4>
      </annotation>

      <annotation annotation_4>
        <belongs_to action action_A>
        <name Brush_Description>
        <composition
           <name Brush_size> <value 2>
           <name Brush_quality> <value Extra-fine>
           <name Brand details>
             <composition
               <name Make> <value 'Super Brushes'>
               <name Model> <value Sublime_brush>
               <name type> <value Size_3>
             </composition>
        </composition>
      </annotation>

    </workflow>
    </brawl>
```

A third example; this example is about an Annotation that contains a link.

Example 3: link-values

What?

An annotation can contain a link reference if the actual does not fit in an ASCII context. This may be the case for media objects like audio, video or graphics.

Explanation

We left out the context of the annotation in this example. Suppose the artefact X in example 2 is the result of the work of a painter (using a very specific brush). In that case we may want to include a reference to the produced object instead of an ASCII-encoded value.

The value in the Annotation uses a link-value; the link itself is a string (presumably) containing a reference to an object outside the scope of BRAWL.

BRAWL

```
…
<annotation annotation_xyz>
  <belongs_to artefact artefact_X>
  <name Work of Art>
  <value link: "http://babbersmolen.files.wordpress.com/2011/09/de-nachtwacht-
rembrandt-van-rijn.gif">
</annotation>
…
```

Scripts

This section contains a few examples of Annotations containing scripts.

Example 4: Simple arithmetics

What?

This example shows the use of simple arithmetic operators in a script

Explanation

Suppose we have an action Xyzzy, that has a context dependent attribute alpha. In the example, annotation_init attributes alpha to action Xyzzy, with a (initial) value of 5.

In annotation_1 a script named 'calc_value' is attributed to Xyzzy. The script contains a few noteworthy aspects. The script contains a single assignment-operation that assigns a new value to the attribute alpha of Action Xyzzy. It uses a dotted reference to this named-value; note that since the Annotation annotation_init attributes this name-value alpha to Action Xyzzy, it is referred to as an element of action_xyzzy instead of an element of annotation_init. Annotations cannot be referred to in BRAWL.

Evaluation of the script, assuming no other scripts have changed the value of alpha, will set the value of alpha to 0. A second evaluation will yield the value -1.25, which BRAWL cannot represent and will be rounded to -1 (assuming BRAWL uses rounding towards zero).

BRAWL

```
…
<action action_xyzzy>
…
</action>

<annotation annotation_init>
  <belongs-to action action_xyzzy>
  <name alpha>
  <value 5>
</annotation>

<annotation annotation_1>
<belongs_to action action_xyzzy>

<name calc_value>
<value
 <script>
  [action action_xyzzy].alpha :=
     ( 2*[action action_xyzzy].alpha + 6) / 8) – 2;
 </script>
> <comment Mind the closing bracket for value!!>

</annotation>
…
```

Example 5: A real script

What?

A first script in a BRAWL Annotation.

Explanation

Suppose we have an Action Xyzzy, that changes behavior depending on the value of (one of) its attributes, alpha. It uses a script decide_behavior to decide what behavior is to be used. The actual behavior is reflected in attribute behavior of Xyzzy.

So in the script decide behavior, the value of alpha is projected onto the range {0…4} using a modulo-calculation. Since BRAWL does not have a modulo-operator, we have to compose it using integer arithmetic operators BRAWL does have. Note that we load this value in the local variable $x.
For the sake of this example the script uses both the if-then-else and the condition conditional statements. The if-then-else construct is used to decide whether the variable $x is in the permitted range, and, if not, sets the behavior to the special error-behavior of Xyzzy.
The condition iterates over the listed conditions until it hits a true condition (that is: a condition yielding TRUE). It executes the corresponding s-expression; in this case they all just set the value of the attribute behavior of Xyzzy.

A more compact, equivalent version of the decide_behavior is given in annotation_2. There the if-then-

else construct has been replaced by a single line in the condition construct. This time the script make use of the way the condition expression is evaluated, top-down until a true condition is encountered, and places an always true condition (namely TRUE) that will cause the error behavior to be selected if all other conditions failed.

Note that the conditions in the condition expression are mutually independent, and each clause could have contained a condition involving variables differing from the others. This means that something like:

```
 condition
    (roses == red)     : do_something;
    (violets == blue)  : do_something_else;
    (grass == green)   : do_nothing;
    TRUE               : do_error_exception;
  end;
```

is completely legal in BRAWL (and could make a lot of sense).

BRAWL

```
…
<action action_xyzzy>
…
</action>

<annotation annotation_init>
  <belongs-to action action_xyzzy>
  <name alpha>
  <value 5>
</annotation>

<annotation annotation_init2>
  <belongs-to action action_xyzzy>
  <name behavior>
  <value "normal">
</annotation>

<annotation annotation_1>
<belongs_to action action_xyzzy>

<name decide_behavior>
<value
 <script>
  begin
     $x := [action action_xyzzy].alpha;
     $x := $x - 5*($x / 5) ;

     if (($x > 4) or ($x < 0))
       then
         [action action_xyzzy].behavior := "error";
       else
        condition
          ($x == 0) : [action action_xyzzy].behavior := "normal";
          ($x == 1) : [action action_xyzzy].behavior := "relaxed";
          ($x == 2) : [action action_xyzzy].behavior := "paranoid";
          ($x == 3) : [action action_xyzzy].behavior := "funky";
          ($x == 4) : [action action_xyzzy].behavior := "lazy";
        end; <comment end of condition>
     end;    <comment end of if-then-else>
  end        <comment end of sequence>
 </script>
> <comment Mind the closing bracket for value!!>

</annotation>

<annotation annotation_2>
<belongs_to action action_xyzzy>

<name decide_behavior2>
```

```
    <comment Improved version of decide_behavior>
    <value
     <script>
      begin
        $x := [action action_xyzzy].alpha;
        $x := $x - 5*($x / 5) ;

        condition
          ($x == 0) : [action action_xyzzy].behavior := "normal";
          ($x == 1) : [action action_xyzzy].behavior := "relaxed";
          ($x == 2) : [action action_xyzzy].behavior := "paranoid";
          ($x == 3) : [action action_xyzzy].behavior := "funky";
          ($x == 4) : [action action_xyzzy].behavior := "lazy";
          TRUE      : [action action_xyzzy].behavior := "error";
        end;    <comment end of condition>
      end       <comment end of sequence>
     </script>
    >           <comment Closing bracket for value>

    </annotation>

    …
```

Example 6: Script complication

What?

Now suppose we want to do something more complex (involving multiple statements) and return a value based on the type of behavior we chose for Xyzzy. Here is how to do it. This shows how we can use the script capabilities of BRAWL for simple programming.

Explanation

Extending the previous example, we defer the operations resulting from the decide_behavior script (also) to other scripts.

The first conditional clause ($x==0) now contains a block that can contain many statements.

The other behaviors are defined in other annotations; note that there is no parameterization and hence no brackets for the 'function calls'. Nevertheless, one script can call yet another script, as demonstrated in the script behave_paranoid.

BRAWL

```
    …
    <action action_xyzzy>
    …
    </action>

    <annotation annotation_init>
      <belongs-to action action_xyzzy>
      <name alpha>  <value 5>
    </annotation>

    <annotation annotation_init2>
      <belongs-to action action_xyzzy>
      <name behavior>  <value "normal">
    </annotation>

    <annotation annotation_1>
    <belongs_to action action_xyzzy>

    <name decide_behavior>
    <value
     <script>
      begin
        $x := [action action_xyzzy].alpha;
        $x := $x - 5*($x / 5) ;

        condition
          ($x == 0) :
            begin
```

```
              [action action_xyzzy].behavior := "normal";
              return 2; <comment 2 is a normal number>
            end;
          ($x == 1) : [action action_xyzzy].behave_relaxed";
          ($x == 2) : [action action_xyzzy].behave_paranoid";
          ($x == 3) : [action action_xyzzy].behave_funky;
          ($x == 4) : [action action_xyzzy].behave_lazy";
          TRUE      : [action action_xyzzy].behavior := "error";
        end;    <comment end of condition>
        return 0;
      end       <comment end of sequence>
     </script>
    >           <comment Closing bracket for value>

    </annotation>

    <annotation annotation_2>
    <belongs_to action action_xyzzy>

    <name behave_funky>
    <value
     <script>
      begin
        [action action_xyzzy].behavior := "funky";
        return 7; <comment 7 is a funky number>
      end       <comment end of sequence>
     </script>
    >           <comment Closing bracket for value>

    </annotation>

    <annotation annotation_3>
    <belongs_to action action_xyzzy>

    <name behave_lazy>
    <value
     <script>
      begin
        [action action_xyzzy].behavior := "lazy";
        return 1; <comment 1 is a lazy number>
      end       <comment end of sequence>
     </script>
    >           <comment Closing bracket for value>

    </annotation>

    <annotation annotation_4>
    <belongs_to action action_xyzzy>

    <name behave_paranoid>
    <value
     <script>
      begin
        $y := ([action action_xyzzy].behave_funky+
               [action action_xyzzy].behave_lazy);
        [action action_xyzzy].behavior := "paranoid";
        return $y;
      end       <comment end of sequence>
     </script>
    >           <comment Closing bracket for value>

    </annotation>

    <annotation annotation_5>
    <belongs_to action action_xyzzy>

    <name behave_relaxed>
    <value
```

```
      <script>
       begin
         $y := ([action action_xyzzy].behave_funky+
                [action action_xyzzy].behave_lazy);
         [action action_xyzzy].behavior := "paranoid";
         return $y;
       end       <comment end of sequence>
      </script>
     >           <comment Closing bracket for value>

     </annotation>

     …
```

9.2.2 BRAWL QoS extension

```
quality_description ::=
      <quality_description>

        <quality>
          quality_name
        </quality>

        <relation>
          comparison_operator  /* <, <=, ==, >=, > or != */
        </relation>

        <value>
            value
          | "[" value "..." value "]"
            /* start- and end-value of same type */
          | "{" value {"," value} "}"
        </value>

        <unit>
          unit_name
        </unit>

       [
        <tolerance>
          value
          /* value of same type as used in <value> */
        </tolerance>
       ]

       [
        <dependency>
          quality_name
          correlation_function
        </dependency>
       ]*

      </quality_description>



quality_name ::=
      service_level |
      quality_requirement|
      integrity_constraint |
```

service_agreement	
Quality type	**Description**
service_level	A specification of the attainable service level of a resource, used to specify the QoS attributes of a BRIDGE registered Resource, allocated to a task in the workflow.
quality_requirement	A requirement on an Action or an Artefact, included in a generated workflow. The requirement originates from a workflow generator's internal knowledge, a participant's deployment policy or a BRIDGE strategy.
integrity_constraint	A constraint on the deployment of a resource, indicating a limit in its capabilities or a boundary in the deployment rules for that resource. The constraint originates from the resource description and resides in the BRIDGE Quality of Service Repository
service_agreement	A contract between a resource allocated to an Action and the BRIDGE CWFGM system. The agreement specifies (an aspect) a level of service to be provided, and is based on the quality requirements of the Action or the Artefacts it produces and the integrity constraints of the resource.
unit_name ::=	
see Table 3	

Domain	Quality/Aspect	unit	Notation	Format
Time				
	Time	seconds	S	float
	Date			YYYY.MM.DD
	Time of day		ToD DToD	hh.mm.ss YYYY.MM.DD.hh.mm.ss
	Duration	seconds	duration	float
	TimeFrame		TimeFrame	{hh.mm.ss, hh.mm.ss} {YYYY.MM.DD.hh.mm.ss, YYYY.MM.DD.hh.mm.ss}
	Frequency	Hertz	Hz, U/s	float
Space				
	Location	latitude, longitude	lat, lng	{float, float}

	Area			{
				{float, float}*
				}

	Distance	meters	m	float
	Range	meters	m	float
	Direction	degrees	deg	float
Physical Environment				
	Dimensions	length width height depth	m m m m	float float float float
	Speed	meters/second	m/s	float
	Temperature	degrees Celsius	C	float
	Weight	grams	g	float
Data				

	Transfer rate	Bytes/second	B/s	number
	Security	levels		open, encrypted
	Privacy	levels		public, restricted (clearance level), private
Resource				
	Status	levels		available, busy, waiting, needs_input, needs_resource, error, off-line
	Skill level	levels		(user-defined)
	Throughput	seconds	s	integer
	Clearance	levels		integer

Table 3: Qualities, Units and Values

9.2.3 BRAWL SLA extension

```
service_level_agreement ::=
      <service_level_agreement>
        <sla_id>
           SLA_ID
        <sla_id>

        <agreement_context>

           <service_requester>
              organisation_descriptor

           </service_requester>

           <service_provider>
              organisation_descriptor
           </service_provider>

        </agreement_context>

        <service_description>
           <capability>
               CapabilityDescription
            </capability>
        </service_description>


          <agreement_guarantee_terms>
            <quality_description>
            { <quality_description> }

          </agreement_guarantee_terms>

      </service_level_agreement>
```

In this syntax `SLA_ID` is a BRIDGE unique identifier, specified by the WF Execution process.

For the BRIDGE project the following aspects for SLA are identified. A SLA specified as an agreement between two organisations:

- The `service_requester` identifies the agency (***organisation_descriptor***) which acts as the requester of the service. It is in the interest of this agency that the service is provided at the QoS as specified.
- The `service_provider` identifies the agency (***organisation_descriptor***) which acts as the provider of the service. It is the responsibility of this agency that the service is provided at the QoS as specified.
- The service description is specified as a `<capability>`-entry `CapabilityDescription`, referring to a capability registered for the resource in the BRIDGE Service Catalogue. The agreement terms are associated to this specific capability.
- The `<agreement_guarantee_terms>` are specified as a quality description, following the syntax specified in section 9.2.2, in which ***quality_name*** is instantiated with the quality type `service_level`.

9.2.4 BRAWL Policy extension

```
policy ::=
       <policy_rule>

         <policy_id>
           POLICY_ID
         </policy_id>

         <policy_name>
           name
         </policy_name>

         (
          <policy_type>
            integrity_constraint
          </policy_type>

          <rules>
            <applies_to>
              {
               resource_descriptor
              }
             | organisation_descriptor
            </applies_to>

            quality_description

          </rules>
         )

         |

         (
          <policy_type>
```

```
            rule_of_engagement
          </policy_type>

          <rules>
            <applies_to>
              {
               organisation_descriptor
              }
            </applies_to>

            <condition>
             quality_description
            </condition>

            <effective>
              {
               quality_description
              }
            </effective>

            [
             <effectuates_integrity_constraints>
              {
               POLICY_ID /* identifying integrity constraint */
              }
             </effectuates_integrity_constraints>
            ]
          </rules>
         )

         |

         (
          <policy_type>
           escalation
          </policy_type>

          <rules>
            <current_situation>
               quality_description
            </current_situation>

            <result_situation>
              {
               quality_description
              }
            </result_situation>

            <authorized_by>
              {
               resource_descriptor /*identifying some authority */
              }
            </authorized_by>

        </rules>
        )

      </policy_rule>
```

In this syntax `POLICY_ID` is a BRIDGE unique identifier, specified by the participating agencies. The `<policy_name>`-entry contains a human readable name for the policy

rule, for example “scale_up_to_minor_crisis”.

For the BRIDGE project we identify three policy types, `integrity_constraint`, `rule_of_engagement` and `escalation`. The rule-format is slightly different for each of these policy-types:

- For the `integrity_constraint` -type policy, the rules section identifies the resources (***`resource_descriptor`***) and agencies (***`organisation_descriptor`***) that the policy-rule applies to and describes a constraint that holds for all affected resources and agencies.
- For the `rule_of_engagement`-type policy the rules section identifies the agencies (***`organisation_descriptor`***) that the policy-rule applies to and specifies a condition `<condition>` that makes the policy rule valid. That implies, that if the condition holds, all affected resources will adhere to the behaviours implied by the `<effective>`-entry.
  For example, if the policy condition (crisis_level == 0) holds, the effective agency policy could state that (deployment_range <= 50 km) and the (operation_level == “local-operation”); the behaviour of all affected resources (all resources of the organisation) will limit their deployment range to 50 km, and apply all rules required by “local-operation”.
  Another rule-of-engagement policy rule might state that if (crisis_level ==1) , the effective (deployment_range <= 150 km) and the (operation_level == “regional-operation”); if the crisis_level gets raised to level 1 (see escalation policy-type), the behaviour of all affected resources (all resources of the agency) will extend their deployment range to 150 km, and apply all rules required by “regional-operation”. In this example, “regional-operation” and “local-operation” are local (agency-wide) variables that can be used in the specification of integrity constraints and (other) rules of engagement.
  The optional `<effectuates_integrity_constraints>`-entry is used to identify (additional) integrity constraints that become effective if the policy condition holds, thus allowing to include, exclude or further detail the behaviour of specific resources. For example, using this option, some resources might remain reserved for local-operation, in spite of the raising crisis level.
- For the `escalation` -type policy, the rules describe a current and (desired) resulting situation. This situation is a global acknowledged variable like the Dutch GRIP-level. Also a list of resources is specified that are authorized to invoke the escalation policy rule.

All policies are to be published (persistently) using the BRIDGE Middleware Publish/Subscribe service in topic “`Apps.Global.WFControl.Policies`”; if the need arises the topic can be partitioned in sub-topics “`escalation`”, “`rules_of_engagement`” and “`integrity_constraints`”.

Policy variables

The policy rules make use of global variables to check whether policy conditions hold. These policy variables are to be defined by end-users.
Policy variables shall be published (persistently) using the BRIDGE Publish/Subscribe

service in topic "`Apps.Global.WFControl.PolicyVariables`"; policy variables shall adhere to the syntax:

```
<policy_variable>
  name-value
</policy_variable>
```

The syntax of **`name-value`** is specified in section 9.2.1.

9.2.5 COMPASS Syntax for Monitoring Recipes

Monitoring Recipes: Syntax and Example

1. Monitors have a number of predefined internal state variables:
   a) State, which has a value "NotStarted", "Started" or "Finished". In the life-cycle of the Monitoring Agent, the value of this variable transitions from "NotStarted" to "Started" to "Finished".
   b) Timer, a timer started at the transition "NotStarted" to "Started" and stopped at the transition "Started" to "Finished" of the state variable. This timer is used to calculate elapsed and remaining times for the task.
   c) Error, indicating whether some error has occurred, and can have the values "TRUE" (initial value) and "FALSE". Error is set from "FALSE" to "TRUE" by the actions associated to conditions, whereas the variable might be checked in the conditions themselves.
2. Conditions contain parameters that have a recipe-unique id (pid). This identifier value is referenced in both actions and conditions. The values of these parameters are either
   a) Stated in the recipe (literal values, such as "`<tod>2015:05:01:12:00:00</tod>`")
   b) Periodically determined by the monitor (such as current time of day or sampling of internal variables)
   c) Periodically retrieved from a repository (such as resource's deployment status)
   d) Extracted from data that is retrieved for a specific subscription (such as subscription to "`App.Global.QualityofService.Triggers`" waiting for a relevant progress trigger.
3. Actions describe
   a) A trigger (message) to be published.
   b) An action to be performed by the monitor (other than publishing a trigger-message). The set of possible actions is limited to:
      i. Setting an internal variable (state or error) to one of the predefined values.
      ii. Starting the internal timer.
      iii. Terminating the Agent.

Syntax specification

```
recipe ::=
    "<monitor_recipe>"
          context_info
          {entry}
    "</monitor_recipe>"

context_info ::=
    "<context_information>"
          "<workflow_id>" identifier "<workflow_id>"
          "<component_id>" identifier "</component_id>"
          "<service_level_agreement>" identifier "</service_level_agreement>"
          "<service_provider>" BRIDGE_ID "</service_provider>"
          "<service_client>" BRIDGE_ID "</service_client>"
    "</context_information>"
```

```
entry ::=
    "<entry>"
          condition
          {action}
    "</entry>"

condition ::=
    "<condition>"
          [ operand ]
          operator      // if previous option absent,
                        // unary operator "-", "mod", "abs" or "NOT"
          operand
    "</condition>"
operand ::=
      condition
    | parameter

parameter ::=
    "<parameter>"
          pid           // In case no option follows, this is a
                        // reference to a previously defined
                        // parameter.
          [ value
          | time_ref
          | resource_ref
          | variable_ref
          | internal_ref
          ]
    "</parameter>"

pid ::=
    "<pid>"
          identifier    // integer!!
    "</pid>"

value ::=               // a literal value
    "<value>"
            integer_val
          | boolean_val
          | string_val
          | time_val
          | location_val
    "</value>"

integer_val ::=
    "<integer>" integer "</integer>"

boolean_val ::=
    "<boolean>" boolean "</boolean>"

string_val ::=
    "<string>" string "</string>"

time_val ::=
    "<tod>" time_of_day </tod>"

location_val ::=
    "<location>"
          "Lat" integer "." integer
          "Lon" integer "." integer
    "</location>"

time_ref ::=
    "<time_ref>"
          time_val
          "current"
          "spent" identifier        // identifier = pid
    "</time_ref>"
```

```
resource_ref ::=
    "<resource_ref>"
          "<id>" identifier "</id>" // resource id
          aspect
    "</resource_ref>"

aspect ::=
    "<aspect>"
           "last_update"           // time value
         | "status" status         // string value
         | "location"              // location value
         | "activity"              // task-id
    "</aspect>"

status ::=
      "busy"
    | "waiting"
    | ...                          // see D07.4

variable_ref ::=                   // Retrieve with subscription
    "<variable_ref>"
          "<topic>" string "</topic>"
          "<key>" string "</key>"          // key-field name
          "<keyval>" string "</keyval>"    // target key-value
          "<field>" string "</field>"      // name of desired value field
          "<type>" type "</type>"          // type of desired field
    "</variable_ref>"

type ::=
      "integer"
    | "boolean"
    | "string"
    | "time"
    | "location"

internal_ref ::=
    "<internal>"
                "state"                  // Monitor internal state values:
                                   // NotStarted
                                   // Started
                                   // Finished
         | "timer"                 // started at transition NotStarted->Started
                                   // stopped at transition Started->Finished
         | "error"                 // TRUE or FALSE
    "</internal>"

operator ::
    "<operator>" operator2 "</operator>

operator2 ::=
      "<" | "<=" | "==" | "!=" | ">=" | ">"
    | "+" | "-" | "*" | "/" | "mod" | "abs"
    | "AND" | "OR" | "NOT"
    | "distance"

action ::=
    "<action>"
          trigger
        | perform
    "</action>"

trigger ::=
    "<trigger>"
          "<topic>" string "</topic>"
          "<type>" triggertype "</type>"
          "<component>" string "</component>"
          "<required>" string "</required>"            // pid
```

```
            "<observed>" string "</observed>"            // pid
      "</trigger>"

   perform ::=
      "<perform>"
            "<performative>"
               performative
            "</performative>"
            {aparm}
      "</perform>"

   performative ::=
        "exit"
      | "start_timer"                  // parameter = timer_id- pid
      | "set_state"                    // parameter =
                                       //    "Started" | "Finished"
      | "set_error"                    // internal var error -> true

   aparm ::=
        "<parm>"
           name-value
   "</parm>"
```

Example

In this subsection, an example is given of a recipe prescribing that (ignoring the recipe context):

1. In case the service providing resource has not switched its state to “busy” after the starting time (2015:05:01:12:00:00) to send out WF_Failure_Trigger (containing the relevant information for mitigation);
2. In case the monitor’s internal state is set to “Finished”, to send out a “WF_Progress_Trigger” (kicking off the next phase if applicable) and to terminate the Monitor Agent (since it’s job is done).

```
<monitor_recipe>

  <context_information>
    <workflow_id>workflow_CPS_WF_1</workflow_id>
    <component_id>action_CPS_2</component_id>
    <service_level_agreement>SLA_12</service_level_agreement>
    <service_provider>0.0.0.69579136426198181 2</service_provider>
    <service_client>0.0.0.9044760271030358294</service_client>
  </context_information>

  <entry>

    <condition>

      <condition>
        <parameter>
     <pid>1</pid>
     <resource_ref>
       <id>0.0.0.69579136426198181 2</id>
       <aspect>status</aspect>
     </resource_ref>
   </parameter>
   <operator> != </operator>
   <parameter>
     <pid>2</pid>
     <value>
```

```
          <string>busy</string>
        </value>
      </parameter>
        </condition>

        <operator> AND </operator>

        <condition>
          <parameter>
        <pid>3</pid>
        <time_ref>current</time_ref>
      </parameter>
      <operator> >= </operator>
      <parameter>
        <pid>4</pid>
        <value>
          <tod>2015:05:01:12:00:00</tod>
        </value>
      </parameter>
        </condition>

      </condition>

      <action>
        <trigger>
          <topic>App.Global.QualityofService.Triggers</topic>
      <type>WF_Failure_trigger</type>
      <component>@context.service-provider</component>
      <required>2</required>
      <observed>1</observed>
        </trigger>
      </action>

    </entry>


    <entry>

      <condition>
        <parameter>
          <pid>25</pid>
      <internal>state</internal>
        </parameter>
        <operator>==</operator>
        <parameter>
          <pid>26</pid>
      <value>
        <string>Finished</string>
      </value>
        </parameter>
      </condition>

      <action>
        <trigger>
          <topic>App.Global.QualityofService.Triggers</topic>
      <type> WF_Progress_trigger </type>
```

```
      <component>@context.component-id</component>
      <required>26</required>
      <observed>25</observed>
         </trigger>
       </action>

       <action>
         <perform>
           <performative>exit</performative>
         </perform>
       </action>

     </entry>

</monitor_recipe>
```