

A FRAMEWORK FOR EVALUATING PROCESS MODELLING LANGUAGES FOR DISTRIBUTED ENVIRONMENTS

ABSTRACT

The paper presents a framework for evaluating how well process modelling languages are suited for representing processes in distributed environments. The framework focuses on five main areas: Distribution, autonomy, heterogeneity, collaboration, and flexibility/evolution. The framework defines 21 quality metrics in the five main areas used to assess the ability to model processes in distributed environments. This paper also presents an evaluation of the four process modelling languages Oz, CAGIS, InConcert, and SPEM using the framework. The evaluation illustrates how the framework can be used and what results are typically found.

Keywords: Process Modelling Languages, Distributed Environments, Evaluation, Quality Metrics

1 INTRODUCTION

It is widely known that there is a strong relationship between software product quality and the software process. Thus, in order to make better software products, the process must also be supported and improved. Process modelling languages (PMLs) represent the conceptual basis for enabling support for analysis, visualization, simulation, communication and enactment of software processes. Most PMLs provide elements for representing processes, activities, products, tools, roles, agents, and relationships between these elements. The introduction of the Internet has provided software companies the opportunity to create software where software teams can be distributed geographically and in time. To be able to support these distributed software processes, there is a need for improved PMLs that can include distribution as a part of the modelling representation.

We have identified five areas to be the main challenge for distributed process modelling. First, the *distribution* itself will affect the process modelling languages in areas like representation of a distributed process, access costs, remote data access, security etc. Second, distribution of a software process will imply that parts of the process must be *autonomous*, and can be managed autonomously. Third, support for distributed processes will also involve management of the *heterogeneity* of tools, process models and representations. Fourth, distributing the software process will require the possibility to model *collaboration* also across organisations. Fifth, a distributed software organisation requires *flexibility* in how the processes can be managed and how they can evolve.

In this paper we present an evaluation framework for PMLs that looks into all issues described above. The framework can be used to evaluate a process centred environment (PCE) being used in an organisation by identifying the shortcomings in PML to be used in a distributed environment. The framework can also be used as a tool to compare several PMLs, e.g. before acquiring a PCE in an organisation.

The paper is organised as follows. Section 2 describes the evaluation framework and its quality metrics. Section 3 describes the four PMLs that are evaluated in the paper. Section 4 describes the evaluation and the results from evaluation of the four PMLs. Section 5 concludes the paper.

2 THE EVALUATION FRAMEWORK

The evaluation framework is a result of analysing several scenarios involving distributed processes, and it was inspired by the Assessment framework for PCEs [3]. Note that all process modelling languages may, in theory, be used in a distributed environment provided that the PCE handles all aspects of distribution. This will inevitably lead to an ad-hoc approach where the PCE must provide the distribution constructs lacking in the PML. The following subsections describe the framework and its quality metrics (QM).

2.1 QM1: DISTRIBUTION

Distribution deals with lower-level support for distributed process model (PM) definition and storage. It refers to distribution among several independent sites and not distribution within a site (i.e. a client/server architecture). A site refers to ([1]) “an administratively cohesive Internet domain sharing a single network file system name space”, e.g. “dept.university.edu”.

2.1.1 QM1.1: Degree of transparency/explicit modelling of distribution

Explicit modelling of distribution is the ability to represent a geographical or network location of a process element in the PM. The transparency concerns whether the information about distribution (e.g., to see that a activity is performed in at another site) is visible to the user in graphical or textual form. No explicit modelling of distribution does not imply that distribution is not supported. As mentioned above, most PMs may be used in an ad-hoc way in a distributed environment if the PCE takes care of the distribu-

tion. The ad-hoc solution is mostly suited for migration of single-site PCEs to multi-site PCEs. The various levels of distribution modelling may be:

1. No explicit modelling of distribution
2. Modelling of distribution that is transparent to the user
3. Modelling of distribution that is not transparent to the user

2.1.2 QM1.2: Dynamic reconfiguration of site network

Dynamic reconfiguration is the ability to change information about distribution in the PM. This metric also describes if such information can be updated automatically or interactively (on user requests). This metric affects to what extent PMs may change location during enactment. The following levels of reconfiguration may be recognised:

1. Static information, no reconfiguration
2. Interactive dynamic reconfiguration
3. Automatic dynamic reconfiguration

2.1.3 QM1.3: Metrics for access time/cost to remote sites

This metric assesses the ability to represent access time/cost parts of PMs that reside on remote sites in a PM. Such information can be used to improve the management of load and remote process elements and avoid long waiting in process enactment because of slow networks or not connection at all. The various forms of metrics may be:

1. No metrics
2. Static metrics for access time
3. Dynamic metrics for access time (updated during enactment)

2.1.4 QM1.4: Data access from distributed sources

This metric assesses the ability to represent distributed data access in the PM enabled by a global name space. In some cases it may just be possible to get information about the remote data, and not actually read or change it, which will limit the possibilities of this feature. The distinct forms of remote data access may be:

1. Data accessed only on the same site
2. Data queried from distributed sources
3. Data queried and read from distributed sources
4. Data queried, read and updated from distributed sources

2.1.5 QM1.5: Security

This metric assesses the ability for the PML to represent access (read, update) restrictions on the PM data itself. The PM may enable modelling of access to specific users, roles or users on a specific site. The following forms of security may be handled:

1. No security
2. User-based security
3. Role-based security

4. Site-based security
5. Combination of user-, role- and site-based security

2.1.6 QM1.6: Security granularity

This metric assesses what parts of sites or PMs that can be modelled with access restrictions. The granularity varies from high-level access like whole sites to low-level access like internal parts of an object (e.g., an attribute or method in the PM). In some cases the creators of a PML has relied on the operating system to provide user-based or site-based security, although this solution is not recommended. The recognised levels of security are:

1. No security
2. Access restrictions handled by the operating system
3. Access restricted to whole or parts of sites
4. Access restricted to whole models
5. Access restricted to specified objects in a model
6. Access restricted to specified internal structures in an object

2.1.7 QM1.7: Activity distribution

This metric assesses whether activities in a PM can be distributed and in what way the distribution is restricted. The activity model may for instance only support either hierarchical decomposition or chaining to distributed sites. The various levels of activity distribution may be:

1. Whole activity network must reside at one location
2. Only hierarchical decomposition to distributed sites
3. Only chaining to distributed sites
4. All activities may be distributed without restrictions

2.1.8 QM1.8: Product distribution

This metric assesses the ability of the PML to represent how products (e.g. source files, documents etc) can be distributed and how product distribution is restricted. The product model may, like the activity model, only support either hierarchical decomposition or product dependencies to distributed sites. The various levels of product distribution may be:

1. Whole product must reside at one location
2. Only hierarchical decomposition to distributed sites
3. Only product dependencies to distributed sites
4. All product parts may be distributed without restrictions

2.2 QM2: AUTONOMY

In the framework we have defined autonomy to be the degree of independence to specify and enact decentralised parts of a PM.

2.2.1 QM2.1: Process model autonomy

This metric assesses to what degree a PM can be represented as a collection of cooperating independent PMs. Autonomy is very important in the cases where a project is divided into several smaller projects, each possibly located

at a single site that wants to have control over their own process. Full autonomy requires that each autonomous PM itself may be distributed and not constrained by the physical location of the process. The levels of autonomy may be:

1. Global PM
2. Global PM with autonomous sub-processes
3. Collection of autonomous PMs constrained by physical location
4. Collection of autonomous PMs

2.2.2 QM2.2: Smallest independent object

This metric assesses what elements in a PM can be addressed as a single independent entity. Usually the activities and the products are the core of the PM, and the other sub models only serve a purpose if related to one of these. This will limit the possibility for full distribution and autonomy among PM objects. The recognised levels of independence are:

1. Whole PM
2. A subset of sub model elements
3. All sub model elements

2.2.3 QM2.3: Schematic autonomy

This metric assesses whether the data definition of PM components is centralised or distributed. To have full autonomy, it is important that the involved actors of each PM have the freedom to create and change their own definitions. The levels of autonomy may be:

1. Global data definition
2. Distributed data definition

2.3 QM3: HETEROGENEITY

In the framework, heterogeneity concerns with the diversity in the ways the PMs and their components are defined, and diversity in the ways they interact with their environment.

2.3.1 QM3.1: Tool heterogeneity

This metric assesses the PML's ability to support tools on different operating systems. This will require that a tool is an abstract and generic entity that may be specialised to concrete, named tools on the various operating systems. It must also be taken into account the differences among different platforms in that some may be script-oriented, like Unix, and others may be user interaction-oriented like MS Windows. The recognised forms of tool heterogeneity are:

1. No tool support
2. Operating system-specific tool support
3. Heterogeneous tool support

2.3.2 QM3.2: Process model heterogeneity

This metric assesses the ability to use more than one PML. Among the heterogeneous PMLs, there is a distinction in whether there is a global interpreter that may convert all PMLs like a "PML virtual machine" or whether the PMLs communicate by standardised agreed-upon interfaces. Non-heterogeneous PMLs may still participate in a heterogeneous environment as long it has a standardised interface to the heterogeneous PML, and the latter act as the glue. The recognised forms of PM heterogeneity are:

1. One common PML
2. One common PML that supports a standardised interface
3. A set of supported PMLs converted to a common PML
4. A standardised interface that each PML must support

2.3.3 QM3.3: Schematic heterogeneity

This metric assesses the ability to use various Data Definition Languages (DDLs) for modelling products. Just like for PMLs, a heterogeneous DDL may either support a set of DDLs that is converted to the common DDL or provide a standardised interface that each DDL must support. Usually non-heterogeneous DDLs already support a standardised interface, like SQL, if the data is stored in a database and may participate in a heterogeneous environment. The recognised forms of schematic heterogeneity are:

1. No associated DDL
2. One common DDL
3. One common DDL that supports a standardised interface
4. A set of supported DDLs converted to a common DDL
5. A standardised interface that each DDL must support

2.4 QM4: COLLABORATION

The framework assesses how collaboration is supported in PMLs by focusing on interaction and information sharing among users and among separate parts of a PM.

2.4.1 QM4.1: Modelling of work context

This metric assesses the ability to represent the work context of the process being enacted in a PM. The work context may be a single common work context for all participants or hierarchically decomposed at several levels in an organisation ranging from the whole organisation to a single user. The work context may also be restricted to a local environment or cross-site borders. The recognised levels of work context modelling are:

1. No modelling of work context
2. Modelling of local single work context
3. Modelling of local hierarchical work context
4. Modelling of distributed single work context
5. Modelling of distributed hierarchical work context

2.4.2 QM4.2: Coordination of distributed process models

This metric assesses the mechanisms for standardized exchange of control information and data between separate parts of a PM. Common protocols may be used to ensure security and independence between the separate parts. Without such protocols there is no distinction between access to local and remote parts of a PM. The recognised ways of coordination are:

1. No distribution of PM parts
2. Coordination of local PM parts through common protocols
3. Distribution of PM parts without coordination
4. Coordination of distributed PM parts through common protocols

2.4.3 QM4.3: Explicit modelling of shared objects

This metric assesses the ability to explicitly model distributed shared objects. The shared objects can be modelled with various forms of access rights for managing simultaneous data access. In some cases negotiation procedures may be used when both exclusive and free access is inappropriate. The various form of object sharing may be:

1. No explicit modelling of shared objects
2. Modelling of shared objects without access control
3. Modelling of shared objects with exclusive access (locking)
4. Modelling of shared objects with free concurrent access
5. Modelling of shared objects with access negotiation procedures

2.4.4 QM4.4: Explicit modelling of distributed group activities

This metric assesses the PML's ability to explicitly represent distributed group activities involving several actors. The PML might have the ability to represent group activities where users work together at the same time or at separate times. The recognised ways of modelling distributed group activities are:

1. No modelling of distributed group activities
2. Modelling of synchronous distributed group activities
3. Modelling of asynchronous distributed group activities
4. Modelling of synchronous and asynchronous distributed group activities

2.5 QM5: FLEXIBILITY AND EVOLUTION

The framework investigates flexibility and evolution by looking at the ability of the involved actors in specifying and enacting the PM.

2.5.1 QM5.1: Degree of user freedom to choose task ordering and execution

This metric assesses the degree of user freedom to choose task ordering and execution, and in what way the PML assigns tasks to users or possibly groups of users. This will

affect the user's ability to have local control of his/her own work. The various degrees of user freedom may be:

1. No freedom, the user may just be assigned to one task at the time
2. The user may be assigned to several tasks and may choose which task to perform
3. A group of users may be assigned to several tasks and each user may choose which task to perform

2.5.2 QM5.2: Specification of incomplete models

This metric assesses what restrictions are put on the PM elements and the relations between them. Strict PMLs will force the user to specify a complete PM for it to be valid. The most usual approach is, like for instance in relational databases that objects being referred have to exist. The following restrictions on model specification are recognised:

1. Each object needs a set of references to other objects, and these objects must be defined before enactment
2. References to other objects do not need to be defined before or during enactment, but all referred objects must be defined
3. References to other objects do not need to be defined before or during enactment, and any referred objects does not need to be defined

2.5.3 QM5.3: Distribution of reusable process model fragments

This metric assesses whether reusable fragments are centrally stored and identified or whether there may be distributed local variants. Local fragment variants may replace more central fragments within the local context. The variants may be local to a site, a group, a single user or a combination. The following means of PM fragment distribution are identified:

1. No modelling of PM fragments
2. Modelling of local fragments only
3. Distribution of fragments
4. Distribution of fragments with site-specific variants
5. Distribution of fragments with group-specific variants
6. Distribution of fragments with user-specific variants
7. Distribution of fragments with a combination of site-, group-, and user-specific variants

3 PROCESS MODELLING LANGUAGES

In this section we present four PMLs that later will be evaluation in Section 4. Two of these PMLs, Oz and CAGIS, represent PMLs from the research community, while InConcert and SPEM represents PMLs used in the industry. Note SPEM is mainly used for description of processes and was not intentionally made for process enactment.

3.1 OZ/AMBER

Oz [2, 1] is a multi-site software engineering environment developed by Colombia University, USA. The main goal of Oz is to ([2]) "support concerted efforts among

geographically-dispersed teams - each local team with its own autonomous process - and emphasize flexibility in the trade-off between collaboration vs. autonomy". Oz is a continuation of Marvel, which was initiated in 1986. Oz was suggested as a geographically distributed and logically decentralised extension of Marvel. Amber has later replaced the Oz PML. Figure 1 shows a graphical representation of a PM in Oz/Amber. The Oz/Amber PML was picked out for the evaluation in this paper because it was designed for distributed process in mind.

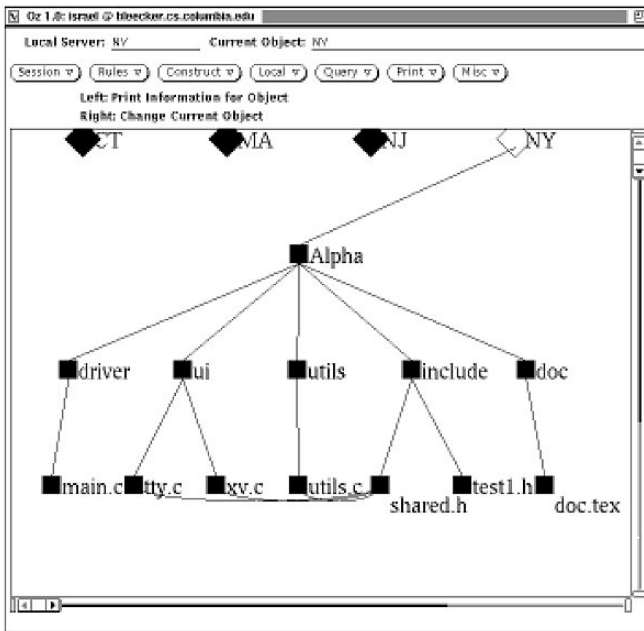


Figure 1. An example of a PM in Oz/Amber

Both Amber and the Oz PML is based on the Marvel Strategy Language, but have added their own extensions. Oz added support for interoperability and collaboration among distributed autonomous processes. Amber, which superseded Oz, stripped away the Oz extensions from the basic language, but was made extensible, so both the Oz and other extensions could be added later. Amber is a rule-based PML where a rule consists of a condition that must be asserted before the rule is executed; an activity that transform the rule parameters and invokes external tools; and a set of effects that hold when the rule has finished. Chaining of rules then determines the flow of control among the activities. Amber's extensibility allows rules to have annotations at specific entry-points in the rule that enables Amber and other rule-based PMLs to exchange messages and data. Both micro-TeamWare and ProcessWeaver have been successfully integrated with Oz using this feature.

Oz supports collaboration between autonomous processes that each may be running on a separate site. Each Oz environment has its own PM, data schema, object base and tools and shares nothing except when explicitly specified. The multi-site extension of Oz uses an *International*

Alliance metaphor for handling interoperability and collaboration. A treaty defines an agreement among processes to share a set of rules. Treaties dictate which sites are allowed to execute the rules, what data is allowed to be accessed, the schema definition of the data and the tool envelopes of the tools intended to be used in the rules. A summit is an execution of a rule defined in a treaty. The summit rule is executed in a single process, but before and after the summit each process may perform its own preparations and assertions. A summit rule may be chained to both local rules and other summit rules.

The product model supports an object-oriented data definition. A class may be specified through multiple inheritance and contains primitive attributes, file attributes (to a "hidden file system" defined in Oz) and 1-to-N-references to other objects. Objects may be decomposed in an aggregate hierarchy. All objects are defined in a local data schema and stored in a local object base.

3.2 CAGIS

The CAGIS PCE is process environment made to support distributed cooperative processes developed at the Norwegian University of Science and Technology. The CAGIS PCE consists of three interacting main components as shown in Figure 2:

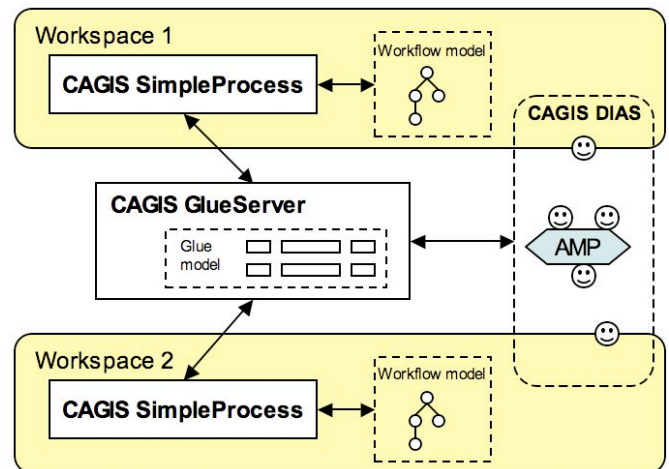


Figure 2. The CAGIS PCE architecture and PML

The **CAGIS SimpleProcess** [6] is a workflow tool with a web-interface, used to model and guide individual activities. An activity can typically be presented to the user as a work description, links to relevant documents or tools, HTML-forms, and/or Java applets. A process in CAGIS SimpleProcess can consist of several autonomous process fragments that can be distributed over several workspaces. To specify the process model, XML or a web-based *process modeller tool* can be used. Hierarchical workspaces are used to model roles in the organization, and CAGIS SimpleProcess has support for moving process fragments between workspaces (roles). The tool uses the *CAGIS SimpleProcess PML* to describe a process as an activity-

network. Each activity is represented by a web page that has a state and relationships to other activities.

The **CAGIS Distributed Intelligent Agent System (DIAS)** [8] provides support for cooperative activities between roles (workspaces). The software agents in CAGIS DIAS can be used to coordinate artefacts and resources, negotiate about artefacts and resources, monitor the working environment for changes or events, provide infrastructure for brainstorming, electronic meetings, trading services etc. CAGIS DIAS provides the infrastructure for creating cooperating agents, and consists of four main components: Agents, agent meeting places (AMPs), workspaces, and repositories. The *DIAS Agent Language* is a high-level agent API written for Java for modelling cooperative activities. KQML is used to define how agents should communicate, and the content of a speech-act (e.g. tell-agent, ask-agent etc.) is specified in XML. It is possible to implement both stationary and mobile agents. Examples of agent methods available are: migrate agent, communicate to other agent, tell other agent, announce service, request service, etc.

The **CAGIS GlueServer** [7] is a middleware that uses a so-called GlueModel, where relations between process fragments (CAGIS SimpleProcess) and software agents (CAGIS DIAS) are defined. The GlueModel can be used to define rules for interaction between roles, and to specify changes of the process model. By using a *separate model* to describe the interconnection between agents and process fragments, it is possible to use other workflow tools as well as other agent systems. This makes the CAGIS PCE open-ended to other systems. The GlueServer uses the *Glue Modelling Language* specified in XML to define the relationship between a process fragment and a software agent, and the reactions to take depending on the results of the agent execution.

3.3 INCONCERT

InConcert was a workflow product developed by InConcert Incorporated, a company spun off from Xerox in 1996. InConcert was used for business and engineering processes and emphasized on ([5]) “extensibility, ease of integration, flexibility and adaptability, and reuse”. The first version of InConcert was released in 1993 after considerable research on using object technology to support workflow processes. We chose the commercial workflow tool InConcert for the evaluation in this paper, because of our experiences with the system. In the evaluation we used version 3.6 of InConcert that was released in July 1997. InConcert is not commercially available any longer.

The InConcert PML is based on Input/Process/Output flow of control, although not in a very strict manner, as processes are not required to have either inputs or outputs. The elements of the PML are tasks, roles and documents. *Tasks* may be hierarchically decomposed into subtasks. The control flow between tasks is made up of signals or triggers, and task dependencies are performed by the tasks guarding their inputs. Signals may, however, not make loops to

support reworking of tasks. A *role* is defined as a pool of people sharing a set of skills. The role is then assigned to a task, either before or during enactment. When assigned, the task is presented to all persons in the pool and any of these may choose to perform it. There is no support for organisational roles, like project manager or sales representative. A *document* is a reference to a data object of any kind that is manipulated by a task. The data may be stored in a relational database or a document management system. During enactment, a running instance of a process is called a Job. Figure 3 shows a screenshot from InConcert 3.6. On the left there is a view of the task hierarchy. On the right there is a view of the task interdependencies. The icons used are chosen by the user and does not correspond to specific kinds of tasks.

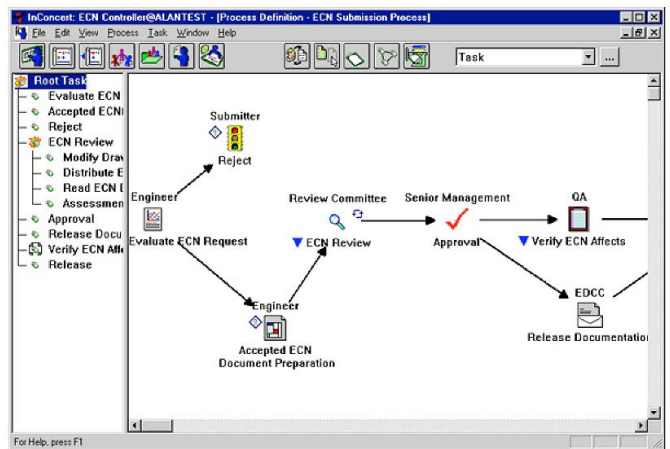


Figure 3. An example of a PM in InConcert

Evolution in InConcert is handled by *process design by discovery*, a feature that lets people record macros that can be stored as workflow rules as they perform their actual work. The recorded process may then be stored in a library of templates and reused. Evolution may take place during enactment, by editing the template or extending the objects themselves by object-oriented inheritance. A template is just a starting point for a new process, so reuse is based on “copy-and-paste” of processes. Extension of objects may only be performed by an administrator, which limits the possibility of letting the users participate in the process and have their own PM fragments.

3.4 THE SOFTWARE PROCESS ENGINEERING METAMODEL SPECIFICATION (SPEM)

SPEM [4] is a metamodel in UML for describing software development processes in UML. Although the main objective of SPEM is to document and describe software development processes for improving human perception of the process, it is possible to specify enactable processes in SPEM. The specification has captured a minimal set of process modelling elements and does not cover specific areas or disciplines like project management and analysis. SPEM

is new PML that has recently started to be supported by various tools and used by researchers and companies. Because of the popularity of UML, it is likely that SPEM will influence how software processes are modelled in the future.

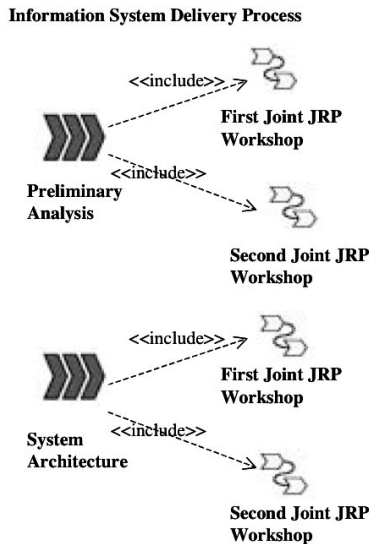


Figure 4. An example of a PM in SPEM

Since SPEM is based on UML, an object-oriented approach for modelling processes is used. The modelling framework consists of four levels: M0-M3. M0 (*performing process*) is used for representing concrete processes as they are enacted in specific projects. M1 (*Process Model*) describes the process according to the template process followed e.g., RUP, independent of a specific project. M2 (*Process Metamodel*) defines the constructs that can be used to specify a PM. Finally, M3 (*MetaObject Facility*) defines how SPEM is structured as a UML profile. As stated above, SPEM can be used to describe general processes as well as specific processes (related to a project). In this paper we will only describe the parts of SPEM that most likely will be used to describe specific processes. A typical process structure in SPEM modelled using the process elements Activity (work performed by one role), WorkProduct (artefact produced, consumed or modified by a process), and ProcessRole (performs activities and is responsible for a set of WorkProducts). At a higher abstraction level, the process can be modelled using the elements WorkDefinition (composite worked performed in a process) and ProcessPerformer (represents performer of WorkDefinitions that cannot be associated with individual roles). Activity is defined as a subclass of WorkDefinition, and ProcessRole as a subclass of ProcessPerformer. The WorkDefinition element can be represented hierarchically, and WorkProducts can be grouped into different types (kinds). In addition, SPEM can represent activity graphs using goals, preconditions, states etc.

4 THE EVALUATION AND RESULTS

In this section we present the results of using the evaluation framework to investigate how well four different PMLs are suited for representing distributed PMs. The tables presented in this section refers to the quality metrics described in Section 2.

4.1 EVALUATING OZ/AMBER

Table 1 presents the results from evaluating the Oz/Amber PML. The Oz PML provides support for explicitly modelling distribution as a part of the PM. Most aspects of QM1 (distribution) are covered but it lacks support for metrics for access time/cost and product distribution. There is also good support for autonomy and to some extent heterogeneity. It is, however, very apparent that Oz is an upgrade of a single site system. Neither activities nor products may be stored distributed, and autonomy is restricted to a single site. The support for defining personal work contexts and personal PM variants is very weak. The biggest strength of Oz is the support for autonomy, and its “shared nothing”-principle also has a number of advantages. There is full local control over the local part of the PM and its products. The summit and treaty protocols gives exact control over how collaboration is performed and data is shared between several processes. The extensibility of the Amber PML allows Oz to act as an umbrella system over several different independent and decentralised PCEs.

Qual. Metric	Score	Qual. Metric	Score
1.1 Expl. distr.	3	3.1 Tool hetr.	2
1.2 Dyn. reconf.	3	3.2 PM hetr.	4
1.3 Access cost	1	3.3 Schema hetr.	2
1.4 Distr. access	4	4.1 Mod. context	2
1.5 Security	4	4.2 Distr. PM	4
1.6 Sec. granl.	5	4.3 Shared obj.	5
1.7 Act. distr.	3	4.4 Group activity	1
1.8 Prod. distr.	1	5.1 Flexible exec.	3
2.1 PM autonomy	3	5.2 Incompl. modl.	2
2.2 Indep. obj.	3	5.3 Reuse pm-frag.	1
2.3 Schema auton.	2		

Table 1. Scores of OZ/Amber from Evaluation

4.2 EVALUATING CAGIS

Table 2 presents the results from evaluating the CAGIS PML. The CAGIS PML has the ability to model distribution of both activities and products. Further, CAGIS PML has strong support for autonomy and heterogeneity of various process elements like activities and products. The autonomy is supported by allowing the individual processes be configured by the user himself. The heterogeneity is supported through the GlueModel, allowing various PMLs to be federated and interact. Further, the CAGIS PML is the only PML in the evaluation with native support for cooperative activities involving several actors accessing the same products. However, the CAGIS PML is weak on security,

access costs and modelling of products.

Qual. Metric	Score	Qual. Metric	Score
1.1 Expl. distr.	3	3.1 Tool hetr.	3
1.2 Dyn. reconf.	2	3.2 PM hetr.	4
1.3 Access cost	1	3.3 Schema hetr.	1
1.4 Distr. access	4	4.1 Mod. context	5
1.5 Security	1	4.2 Distr. PM	4
1.6 Sec. granl.	1	4.3 Shared obj.	5
1.7 Act. distr.	4	4.4 Group activity	5
1.8 Prod. distr.	4	5.1 Flexible exec.	2
2.1 PM autonomy	4	5.2 Incompl. modl.	3
2.2 Indep. obj.	3	5.3 Reuse pm-frag.	3
2.3 Schema auton.	2		

Table 2. Scores of CAGIS from Evaluation

4.3 EVALUATING INCONCERT

The results from evaluating InConcert are shown in Table 3. InConcert does not perform very well against the quality framework. There is little support for distribution and autonomy and only partial support for heterogeneity. Distribution support was promised, but the product has now aborted. The biggest flaw of InConcert is the fact that tasks may not have dependencies backwards in the task chain, which means that processes involving feedback and reworking of tasks cannot be modelled properly. The strengths of InConcert are flexibility and evolution. The *process design by discovery* concept is a very intuitive way of creating new PMs and handling exceptions in existing models during enactment. A nice flexibility feature is the assignment of a pool of users to a task and any user may choose to perform it. This gives a sense of control that should not be underestimated. Another big strength of InConcert is that it is a commercial product that is tested thoroughly and installed in a large number of organisations. It has a better user interface and stronger security features than most university prototypes, as these are features that need to be in a commercial product, but are not necessarily scientifically interesting.

4.4 EVALUATING SPEM

The results from evaluating SPEM are shown in Table 4. Intentionally, SPEM does not cover specific domains like support for distributed processes. This is reflected in the results with low scores for most metrics in the framework. However, autonomy (QM2) is supported to some extent by allowing the process to be specified into several autonomous sub-processes and that all process elements can be uniquely identified and addressed. There is also some support for heterogeneity (QM3) provided by support for tools and standardised interfaces for PMLs and DDLs. Note that the tool support in SPEM is mainly focusing on process guidance. It is rather surprising to see that SPEM only has limited support for modelling cooperative aspects (QM4) of the software process. It is a bit hard to judge

Qual. Metric	Score	Qual. Metric	Score
1.1 Expl. distr.	1	3.1 Tool hetr.	3
1.2 Dyn. reconf.	NA	3.2 PM hetr.	2
1.3 Access cost	NA	3.3 Schema hetr.	3
1.4 Distr. access	1	4.1 Mod. context	3
1.5 Security	2	4.2 Distr. PM	1
1.6 Sec. granl.	5	4.3 Shared obj.	2
1.7 Act. distr.	1	4.4 Group activity	1
1.8 Prod. distr.	1	5.1 Flexible exec.	3
2.1 PM autonomy	1	5.2 Incompl. modl.	2
2.2 Indep. obj.	2	5.3 Reuse pm-frag.	2
2.3 Schema auton.	1		

Table 3. Scores of InConcert from Evaluation

how well SPEM supports flexibility and evolution (QM5). The SPEM itself makes it possible to allow some flexibility and evolution, but these issues must also be handled in a possible PCE supporting SPEM as a PML.

Qual. Metric	Score	Qual. Metric	Score
1.1 Expl. distr.	1	3.1 Tool hetr.	3
1.2 Dyn. reconf.	NA	3.2 PM hetr.	2
1.3 Access cost	NA	3.3 Schema hetr.	3
1.4 Distr. access	NA	4.1 Mod. context	3
1.5 Security	1	4.2 Distr. PM	2
1.6 Sec. granl.	1	4.3 Shared obj.	1
1.7 Act. distr.	1	4.4 Group activity	1
1.8 Prod. distr.	1	5.1 Flexible exec.	3
2.1 PM autonomy	4	5.2 Incompl. modl.	1
2.2 Indep. obj.	3	5.3 Reuse pm-frag.	2
2.3 Schema auton.	1		

Table 4. Scores of SPEM from Evaluation

4.5 SUMMARY OF THE EVALUATION

From the evaluation of the four PMLs above we clearly can see that there is a great variation in how various PMLs addresses distribution. The evaluation shows, as expected, that Oz/Amber and CAGIS score well as they were design to support distributed environments. The main difference between Oz/Amber and CAGIS, and the other PMLs is the explicit support modelling distribution, a better support for autonomy and support for heterogeneous PMs. InConcert's strength is providing support for heterogeneous tools and schemas. SPEM does not perform particularly well in the evaluation mainly because distribution has not being taken into account in the meta model. Because of the flexibility of UML, it is likely that SPEM can be tailored to represent distribution in a more direct manner.

This evaluation also identified some issues that are not properly addressed in the selected PMLs. No PMLs allow modelling of access time/cost to remote hosts. This problem can also be addressed by using awareness tools that

will monitor links to remote sites. Also only CAGIS PML allows activities to be distributed without restrictions. This issue is also related to the flexibility of the architecture for the PCE. We have also identified that only CAGIS PML allow full distribution of products. The main problem with full distribution of products is to manage product evolution to ensure consistency. This problem has not been properly addressed in the CAGIS PCE. Here the PML must make a trade-off between absolute consistency of data updates and how flexible the users can update their products.

5 CONCLUSIONS

In this paper we have presented an evaluation framework to investigate how well a PML can represent a software process in a distributed environment. The framework evaluates the PMLs in the five areas distribution, autonomy, heterogeneity, collaboration, and flexibility/evolution. Further, the framework identifies several issues that must be taken into account before selecting or creating a PML for a distributed environment. Further, we have presented an evaluation of four PMLs describing how the framework can be used and what results can be expected from it. From this evaluation, we can see that there is still room for improvement for PMLs in order to model and support a distributed software development process. We hope that others find the framework useful and that other PMLs can be analysed using the framework. The framework can also be used to evaluate how well workflow modelling languages can model and represent business processes in a distributed environment.

REFERENCES

- [1] I. Z. Ben-Shaul. Federating Process-Centered Environments: The Oz Experience. *Process Technology: Special Issue of the Journal Automated Software Engineering*, 5(1), 1998.
- [2] I. Z. Ben-Shaul and G. E. Kaiser. A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment. In *Proc. 16th Int'l Conference on Software Engineering, Sorrento, Italia*, pages 179–190, May 1994.
- [3] J.-C. Derniame, B. A. Baba, and D. Wastell. *Software Process: Principles, Methodology, and Technology*. Springer Verlag LNCS 1500, Berlin, Germany, 1998.
- [4] Object Management Group. *Software Process Engineering Metamodel, Version 1.1*, January 1 2005. Web: <http://www.omg.org/technology/documents/formal/spem.htm>.
- [5] S. K. Sarin. Object-Orient Workflow Technology in Concert. In *Forty-First IEEE Computer Society International Conference: Technologies for the Information Superhighway*, pages 446–450, Santa Clara, California, February 25-28 1996.
- [6] A. I. Wang. Support for Mobile Software Processes in CAGIS. In R. Conradi, editor, *Seventh European Workshop on Software Process Technology*, Kaprun near Salzburg, Austria, 22-25 February 2000.
- [7] A. I. Wang, R. Conradi, and C. Liu. Integrating Workflow with Interacting Agents to support Cooperative Software Engineering. In *Proc. IASTED International Conference Software Engineering and Applications*, Las Vegas, Nevada, USA, 6-9 November 2000.
- [8] A. I. Wang, C. Liu, and R. Conradi. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, 17-19 June 1999.