Requirements for software-support in process model reuse: a systematic literature review and discussion of three state-of-the-art process modeling environments

Research Report

Selim Erol


1 Introduction

Business process orientated management can be regarded a common practice among today’s large and mid-size organizations. As a consequence many organizations maintain a large body of business process documentation. Along with process documentations usually also a large number of process model artifacts are maintained in repositories or process portals to consolidate and share process knowledge among stakeholders of a process. Both from the perspective of process modeling efficiency and the perspective of process model consistency it is highly desirable to encourage the reuse of process models or at least parts or elements of existing process models. Duffy et al. [1995] mention several reasons why reuse might be beneficial to the quality of design processes. Namely, reducing design time and effort, preventing unnecessary variety and proliferation, avoiding duplication of design faults, evolving design solutions, keeping consistency of design solutions are only some of the reasons recurrently outlined in literature.

While activities such as process model creation and formal validation are well understood and extensively supported by state-of-the-art process modeling environments only limited insight in the process of process model reuse and related requirements for efficient software-support exists.

The research effort presented in this paper is mainly targeted at an identification and characterization of relevant activities for process model reuse. The set of identified main activities and their sub-activities serve as a starting point to outline generic requirements for respective software-support. The identification of main activities is based on an extensive review of relevant literature in the field. Based on the derived requirements for software-support a discussion of three state-of-the-art process modeling environments is provided in section.
2 Related work

A study that pursues a similar goal is presented in [Shahzad et al., 2010]. The authors claim that stakeholder requirements for process model repositories have not been sufficiently investigated and therefore conducted a series of interviews and surveys with practitioners and researchers to elicit a set of eight generic requirements for process model repositories to support reuse of process models. The requirements identified correspond largely with the requirements outlined in this paper. However, the research work of Shahzad et al. [2010] differs from the work described in this paper in various ways. First, in this work prior research in the field of knowledge reuse and design artifact reuse has been taken into consideration through a systematic literature review. Second, the primary goal in this work was to develop an understanding of the process of reuse independently of any tool support and subsequently derive generic requirements. An interesting finding in Shahzad et al. [2010] is that practitioners and researchers differ in their perception of some requirements importance. For example, the importance of generic classification schemes to support process model search is more strongly supported by researchers than practitioners. Another example is that practitioners are more neutral when it comes to the requirement of maintaining a domain independent layer of process models to facilitate reuse.

Yan et al. [2012] suggest a business process model repository management model along with a reference architecture. The management model outlines the functionality that should be provided. The reference architecture represents the main components that provide this functionality and their relations with each other. The authors use this framework subsequently to discuss the proposed model in the light of several publicly available process repositories and repository architectures. A future research agenda. A main difference to our research work is that the repository management model suggested (which can be understood as well as a set of requirements for process model repositories) does not explicitly focus on the reuse aspect. However, the work of Yan et al. has some commonalities with our process oriented perspective on reuse. This especially true for the process function model (as a part of the overall repository model) which outlines basic functions on very high level that need to be provided for storage, retrieval and integration of process model. In addition the authors suggest a process reuse architecture that describes the main components and their interactions to enable functions like process model storing and retrieving. Compared with their requirements postulated the authors state a rather large gap to current state-of-the-art process model repositories.

In Markovic and Pereira [2007] several reuse scenarios are outlined and subsequently requirements are derived. The authors use the requirements to suggest an expressive formalism for describing business process models to support reuse in business process modeling. The formalism makes use of π-calculus and several business ontologies to describe processes which is expected to improve effectiveness of process model retrieval and adaption.

In Elias and Johannesson [2012] the authors have outlined several requirements for process model repositories. They have chosen to focus on collections of process models both informal and formal and evaluate their capability for reuse in modeling environments. This is in contrast to the approach presented in this paper where the focus is on design and evaluation of modeling environments in terms of reuse capability. However, the requirements identified served as a major input for this research.
3 The process of reuse

For understanding the process of reuse in the context of process modeling three areas of research have been identified as relevant. First, knowledge reuse is a research area within the domain of knowledge management that has a large body of literature studying the nature of organizational knowledge reuse and the factors affecting effective knowledge reuse. Second, a particular type of knowledge reuse – design (knowledge) reuse – has been widely studied within the field of engineering. Finally, software reuse as an artifact centered type of reuse has been studied as well extensively within the field of software engineering. In the following I will mainly refer to research results from the above mentioned research areas.

The need to support organizations in knowledge management through adequate tools has been recognized as early as in the 1970s [Tampoc, 1993]. However after a first hype and resulting disappointments [Davenport et al., 1998; Storey and Barnett, 2000] in practical application effective knowledge management is still recognized a primary concern in modern organizations [Von Krogh, 2012]. Early conceptualizations of knowledge management (e.g. Nonaka [1991]) have mainly focused on the need to externalize knowledge, as explicit knowledge in contrast to tacit knowledge “is formal and systematic. For this reason, it can be easily communicated and shared, in product specifications or a scientific formula or a computer program” [Nonaka, 1991, p. 98]. Other researchers like Davenport and Prusak [2000] point to the fact that knowledge becomes a ‘valuable corporate asset only if it is accessible, and its value increases with the level of accessibility’. The relevance of knowledge management research for process management in general and process design in particular has been recognized by zur Mühlen [2003] and Adamides and Karacapilidis [2006]. Research work referring explicitly to knowledge reuse has been carried out for example by Markus [2001] and Majchrzak et al. [2004]. Markus developed a theory of knowledge reuse that outlines the basic concepts of reuse. Accordingly, they proclaim the activity of reuse to necessitate several preceding activities: (1) capturing or documenting knowledge, (2) packaging knowledge for reuse, (3) distributing or disseminating knowledge (providing people with access to it), and finally (4) reusing knowledge. The authors state that reuse cannot be seen in isolation but in the context of the process of knowledge creation. Packaging knowledge for reuse refers to “culling, cleaning and polishing, structuring, formatting, or indexing documents against a classification scheme”. Distributing or disseminating knowledge is the act of making knowledge available through active knowledge transfer (e.g. communities-of-practice or learning initiatives) or publishing services or the creation of repositories for users to browse and discover relevant knowledge. The actual reuse activity involves the following sub-activities: (4.1) defining the search question, (4.2) search for, and location of, experts or expertise, (4.3) selection of an appropriate expert or of expert advice from the results of the search, and (4.4) applying the knowledge which the authors describe as well as the process of “recontextualization” of knowledge. Regarding the types of users involved in the knowledge reuse process Markus [2001] makes a distinction into knowledge producers, knowledge consumers and knowledge intermediaries. The latter role is concerned with packaging knowledge for reuse (see above). These roles can be performed by a single individual or by different individuals as well as combinations hereof. Where the individuals performing tasks do not have to be necessarily human agents, e.g. tasks like knowledge classification and dissemination can be performed by software agents.

In addition to Markus often cited process of knowledge reuse Demian and Fruchter [2006] suggest a distinction into internal and external knowledge reuse. Whereas internal knowledge reuse refers to the scenario where knowledge is reused from the individual’s internal memory, in contrast external knowledge reuse refers to the act of reusing knowledge from an external memory (e.g. a knowledge
Furthermore the authors conceive the process of knowledge reuse as a three-step activity: (1) finding a reusable item, (2) exploring its project context in order to understand it and assess its reusability, (3) exploring its evolution history in order to understand it and assess its reusability. A major tenet in this concept of reuse is that the context of a reusable item has two dimensions: project context and evolution history. Each of this contextual dimensions can be explored upward/downward along levels of knowledge granularity or forward/backward along stages of knowledge development to understand and assess it for it’s potential applicability.

The process of design knowledge reuse has been investigated for example by Duffy et al. [1995]. The authors suggest a process model of design reuse that distinguishes between the notions of “design by reuse” which is the actual reuse process, “design for reuse” which is closely related to “design-by-reuse” and “domain exploration”. According to the authors, the design reuse process is a cyclic process that starts with the abstraction of a new design “design for reuse” to complement the existing domain knowledge which is stored in a respective repository both in the form of a reusable design artifact and information relating to the product, process and rationale. Through “domain exploration” a designer uses his domain model to evaluate the reusability of existing design knowledge to create a new design – “design by reuse”. Similar to Demian and Fruchter [2006] also Baxter et al. [2008] argues that effective design knowledge reuse is reached through exploiting both dimensions of context: the knowledge related to past design processes – meaning the evolution of the artifact and related design decisions – and knowledge related to the product/artifact of design itsel. In Johansson and Kliger [2007] a case-study in structural engineering revealed a sequence of activities recurrently taking place in reuse: (1) recalling a previous design solution, (2) finding information about the previous design solution, (3) understanding the previous design solution and (4) copying and adapting information about a previous design solution.

The rather large body of research dealing with software reuse focuses mainly on providing methods, techniques, and tools to support the development of reusable software and components hereof. Only few research studies deal with the process of reuse from a developer’s perspective. However, Mili et al. [1995] state that reuse-centered design will affect the software development process on two levels as both the process of developing for reusable assets and the process of developing with reusable assets need to be considered equivalently. The first aspect – developing for reusable assets – has been addressed by approaches like component, domain engineering and architectural pattern based approaches Shiva and Shala [2007]. The second aspect has been addressed by research in component retrieval, specification and description techniques and languages (see for example Sugumaran and Storey 2003 Gu et al. 2005).

Probably the most elaborate investigation on the activities involved in software reuse is given by Kim and Stohr [1998]. Kim and Stohr similar to Mili et al. suggest that activities can be roughly classified into “producing activities which involve the identification, classification and cataloging of software resources, and consuming activities which involve the retrieval, understanding, modification, and integration of these resources into the software product”. Identification, classification and cataloging are activities referring to the creation and maintenance of a library or repository of reusable assets. Retrieval, understanding, modification, and integration are activities which are interwoven with other software development activities such as requirements specification and building new software resources. Kim and Stohr [1998] point to the importance of considering all sorts of resources that are related to the software development process rather than considering only software components. This line of thought is followed for example by Crnkovic et al. [2006] who suggest to integrate the before mentioned reuse activities into the software development processes. Accordingly, the activities for component design, implementation and testing are conducted in parallel with the
identification of reusable components and their evaluation, selection, adaption, integration. Ye and Fischer [2002] and suggest four different modes of (re)use. Reuse-by-memory mode refers to reusing a piece of knowledge which is residing in user’s memory and can be reused without external help. The reuse-by-recall mode refers to a user having only partial awareness of a piece of knowledge that might help in a particular design problem. A user must consequently reconstruct this partial knowledge either by searching in a repository or by asking other experts. The reuse-by-anticipation mode is a way of reuse that is grounded in the past experience of a designer which makes him anticipate a reusable piece of software without exact knowledge of where and how to search for. Finally, reuse-by-delivery is a mode of reuse that is not actively initiated by the user himself. Rather, reuse is triggered from outside to help users in acquiring reusable assets. Each of this modes implies a different process of reuse activities to be performed in the software design process McCarey et al. [2008].

To summarize the above reuse can be regarded as a set of activities that is embedded into a higher level problem solving activity like design and engineering (or even more general problem solving activities) activities. Reuse can be classified into activities that lead to reusable artifacts (“design-for-reuse”) and those activities that lead to reused artifacts (“design-by-reuse”). For the first class of activities an individual has to make assumptions about future reuse scenarios and has to integrate these assumptions into the artifacts under design. For the second class of activities an individual has to recall, reconstruct or anticipate reusable artifacts and has to integrate these artifacts in to the current context.

4 Conceptualizing the process of process model reuse

In this section the above findings with regard to reuse activities in problem solving processes will be integrated with the process of process modeling as conceptualized in [Frederiks and Van der Weide, 2006; Hahn et al., 2011]. Frederiks and Van der Weide refer to modeling as a process consisting of four main activities: (1) Elicitation, (2) Model building, (3) Validation, (4) Verification. Elicitation is the act of collecting information about the domain of interest from domain experts, verbalization of this information into an initial specification and further reformulation into an informal specification. Modeling is targeted at the transformation of the informal specification into a formal specification that comprises a conceptual model along with grammar rules, constraints and a lexicon. The formalization of the informal domain description is reached through identifying relevant domain concepts and their relations (grammatical analysis) and mapping concepts to model language specific concepts. Validation refers to the act of evaluating the congruence of the formal specification with the informal specification. Through verification the formal specification is checked for internal correctness. The activities involved mentioned above are not conducted in a strictly sequential manner rather they are performed in a repetitive and cyclic way.

In figure 1 the process of process modeling is illustrated in the sense of [Frederiks and Van der Weide, 2006; Hahn et al., 2011] extended with typical reuse targeted activities as discussed in the previous section. In the following this process model will be discussed in detail and in the light of existing theoretical and technological approaches to process model reuse. The modeling activity is detailed in figure 2. Accordingly, two principal courses of action might be taken when modeling. One course is the creation of a new model according to the requirements and informal specification of a process given. This approach which can be called “greenfield approach” or radical approach does not refer to/presume any prior model or part hereof to fit the informal specification. Consequently no reuse is performed. The other course of action is that a search for a reusable model or part hereof
is performed. Usually this course of action is taken when a possibly reusable model (part) is known in advance or at least assumed to exist somewhere. The modeling activities described below can be classified into “modeling-by-reuse” activities and “modeling-for-reuse” activities which are inherent in model creation, model storing, and model annotating activities.

4.1 Process model search (A1)

Searching and finding an adequate process model for reuse is a fundamental activity in any reuse scenario. This becomes even more relevant in organizations where large process model repositories have evolved over time.

For this purpose the first sub-activity to be performed is the transformation of requirements into a search question (A1.1). In other words, the outcome of a preceding requirements elicitation activity (informal specification of domain concepts) is used as the input for the formulation of a search question.

(→R1 A1.1) The search question contains the terms to be searched for and logical relationships with each other (e.g. AND/OR operators). The search question must be articulated through a language that is interpretable by the search agent. A user must be able to formulate a search question explicitly through a query language, e.g. SQL or high-level diagrammatic languages as suggested for example by [Awad, 2007; Beeri et al., 2008; Markovic et al., 2009]), or implicitly where a query is generated from selection of predefined high-level process concepts, e.g. a process classification scheme. A search query may also be generated implicitly from a model currently under construction (thus incomplete) like it is suggested in [Hornung et al., 2008; Markovic et al., 2009; Awad et al., 2011].

(→R2 A1.1) The terms, phrases or concepts used to search for a possibly reusable model are likely to vary with the specific context of a modeling effort. However, a general distinction can be made between those search questions that address the contextual data (e.g. the project where it was used), meta data (e.g. title, creation date, author) of a process model, those that are targeted at the content. The content of a process model is manifested through the classes of model (language) elements used and their concrete instantiations. A user must be able to specify a search question by
Figure 2: Basic activities related to the activity of process model reuse. Distinction between “model-by-reuse” and “model-for-reuse” activities.
referring not only to the content but also by related data such as contextual information and meta data. Furthermore a user should be able to query for a specific behavior of a process model (part) in terms of it’s flow logic, e.g. behavioral pattern [Tran et al., 2011] or profile [Smirnov et al., 2010] of a process model rather than a concrete process model.

In the case process model repositories contain process models described through different modeling languages an abstraction layer is necessary to allow for language independent query specification and retrieval of process models [La Rosa et al., 2011].

The specification of the search space (A1.2) is a necessary sub-activity that has to be performed as well in advance of the actual search activity. In the case of process models the search space is specified by referring to one or more process model repositories. Process model repositories may be located within organizations boundaries or may be provided by externally (e.g. a web service). In practice, process models within organizations boundaries are not only stored in a single centralized repository but are scattered among multiple repositories ranging from simple file system based repositories to specialized model databases.

For a successful and efficient search of process models multiple repositories must be addressable and accessible through a search query. At least a modeler must be able to specify repositories relevant for the search question and perform subsequent search queries. A user should be able to specify a repository independently of the actual repository implementation and location [Yan et al., 2012].

The execution of the search query (A1.3) can be understood as an activity where a set of process models within the search space is checked whether or to which extent it matches a set of search criteria. A search query may return none, a single or multiple candidate process models. The latter case may occur when multiple process models equally satisfy the search criteria.

Retrieval of matching process model content implies that a search query in a process model specific query language needs to be transformed into a database specific query language (e.g. SQL, XQuery). Additionally, the search results of a query need to be interpreted and presented in a way that allows a modeler for efficient evaluation and selection in terms of reusability [Choi et al., 2007; Beeri et al., 2008].

The degree to which a candidate process model matches the search criteria must be quantified through some kind of similarity metric (similarity between search criteria and elements of search space). A threshold value or interval for the similarity metric determines the number of models qualified as search results. Scores for the relevance of candidate process model and subsequent rankings of candidates support a modeler in decisions regarding the potential reuse of a process model. However, similarity metrics for process models must reflect similarity on the textual level – comparing the labels of model elements of both query and candidates in the search space, but also similarity on the semantic level – considering as well the semantics of model elements and similarity on the behavioral level where as well relations between model elements are considered [Koschmider et al., 2010; Awad et al., 2011].

Combinations of similarity measures with measures of model usage frequency and as well subjective rankings from other modelers may support the evaluation of process model relevance [Koschmider et al., 2010].

Related research Based on several usage scenarios [Awad, 2007] identified a set of requirements for a query language targeted at process model retrieval. Accordingly, a respective query language should understandable by non-technical users. It should therefore be based on a particular process modeling language. Furthermore a query language should support the navigation of process structures in a
way that whole process model structures and parts hereof can be searched for. The authors also propose a feature to iteratively refine queries. To fulfill these requirements the authors suggest a query language (BPMN-Q) that is based on BPMN. BPMN-Q is query language that allows to formulate a search query through a diagrammatic representation. Special diagram elements can be used to express typical query operators like wild cards and logical connectors of search terms. In their follow-up work in [Awad et al., 2011] an approach is presented that allows for “modeling-by-selection” where a modeler implicitly – during model creation – submits queries to a process model repository and continuously gets suggestions for model completion based on existing models.

[Beeri et al., 2008] propose a query language that is especially designed to retrieve BPEL based process descriptions from a repository. Through a diagrammatic language that allows to model desired state patterns of a web service a modeler is able to query for candidate web service descriptions. The query language BP-QL is based on an intuitive model of business processes which is an abstraction of the BPEL specification with the advantage to hide tedious XML details. Hence, each BP-QL query is transformed into an XQuery representation which allows to effectively search in XML based BPEL documents.

Similar to Awad [2007] also Markovic et al. [2009] plead for a user-friendly query interface to specify queries which is not too different from commonly used applications by the users or process models. However, in addition to behavioral querying which is based on the flow logic of the process models users may also perform static queries through selection of process related annotations like business goals, functions, roles and resources or the type of artifacts (model, pattern, fragment, guideline).

Ehrig et al. [2007] propose an ontology to describe Petri-net based process models in an unambiguous format that facilitates logical reasoning and the automation of process composition. Based on this ontology and the use of a thesaurus (WordNet[1]) a metric is suggested that allows to compute both syntactic and semantic similarity for the labels and concepts used within business process models.

[van Dongen et al., 2008] suggests an approach towards measuring the behavioral similarity of process models. The authors use the concept of causal footprints to describe the relationships between activities. So called look-back and look-ahead links reflect the relations of each activity with it’s neighboring successor and predecessor activities. These footprints can subsequently be used to compute a footprint vector in analogy to document vectors [Salton et al., 1975] from the domain of information retrieval. The similarity measure is then represented through the cosine of the angle between those vectors. In [Dijkman et al., 2011] the concept of causal footprint similarity is extended with label matching similarity and structural similarity measures. Label matching refers to the comparison of labels attached to process model elements. Structural similarity compares element labels as well as the graph topology of process models.

In [Weidlich et al., 2011] an approach to similarity computation is presented that is based on so called behavioral profiles. Accordingly, a lower level process model can be automatically transformed into a higher level model by deriving a set of relation types between activities (a behavioral profile) and synthesizing these relations by means of reduction rules into higher (more abstract) level representations. Although the approach is mainly targeted at consistency checking between higher level process models (conceptual models) and process models on a technical level it can be potentially adopted for computation of behavioral similarity. A respective metric for behavioral similarity that is based on the above discussed concept has been suggested by Kunze et al. [2011]. The advantage of the behavioral profile approach over the approach presented in [van Dongen et al., 2008] is mainly

---

1 http://wordnet.princeton.edu
that also different types of relations are considered. However, application of this approach requires a process model to satisfy several formal constraints and assumptions that might not relate to practical necessities.

4.2 Process model inspection and selection (A2)

A modeler being presented a set of resulting process models from a precedent search activity needs to inspect and evaluate each model (A2.1) for potential reuse. In a first attempt a modeler might be interested mainly in the information about what the process model does rather than how it does [Tracz, 1991].

(\(\text{→R1}_{A2.1}\)) This requires that the results are presented in a way that the modeler is able to study the model itself, the meta-information attached to the model and contextual information. Information about the context of use must be provided by explicit contextual information provided along with the process model and implicitly by the models relations to other process models. Information about the external behavior of a process model must be provided by a detailed description of the interfaces of a process model, the input required, output produced, goal satisfied. The internal behavior is represented through the flow of activities, the participants, data objects and resources involved [La Rosa et al., 2011; Shahzad et al., 2010].

(\(\text{→R2}_{A2.1}\)) Necessary adaptions must be recognized from the requirements given and subsequently be quantified to allow for an estimation of adaption costs. In a scenario where multiple modelers make use of a process repository also judgments from other modelers should be provided, e.g. information on the reasons of the model’s reuse, by whom it was reused and how often it was reused before [Erol et al., 2010; Koschmider et al., 2010].

In the case where more than one process model has to be inspected for potential reuse, a modeler has to select (decide for) a process model (A2.2) based on the results of prior inspections. Final selection of a process model is primarily a cognitive task that might take into consideration the costs of adapting the model to the new requirements.

(\(\text{→R1}_{A2.2}\)) Deciding and selecting for a particular process model for reuse requires that a user is able to indicate his choice and subsequently transfer the model of choice to a new context, e.g. that the model is copied or referenced to a new project space.

4.3 Process model adaption (A3)

As soon a reusable process model artifact is selected for reuse the process model may need to be adapted for the new context it will be used in. Adaptation can possibly be performed manually or automatically. In both cases new requirements must be transformed into a set of atomic change operations that are applied to the original model to receive a new process model.

(\(\text{→R1}_{A3}\)) Local adaptions of parts of a process model usually need consideration of its global behavior and possible effects. Therefore a modeler might need guidance in evaluating the impact of change and which additional changes have to be applied to satisfy formal correctness, consistency and validity of the new process model. To evaluate the impact of change also a simulation mode or sandbox should be provided that allows for changes without actually taking effect. See also R2_{A2.1}.

(\(\text{→R2}_{A3}\)) In case that changes to process models are frequent and complex also automation of model changes according to a given set of requirements or contextual variables should be possible. At least suggestions for changes would be desirable as it reduces the cognitive load of a modeler in the first run but leaves the authority of final changes to the modeler. This is especially beneficial for
executable process models (workflow definitions) where model elements refer to software components or services [Lee et al., 2007; Hallerbach et al., 2008; Grambow et al., 2010].

Changes must not depreciate the original model. Rather a copy of the original model must be created that is subsequently changed and constitutes a new model for a new context of use. However, a reference to the original model should be kept to be able to trace back changes and be able to compare variants in use [Shahzad et al., 2010].

Related research A particular technique to support adaption of process models for a new context of use has been introduced by Recker et al. [2006] and Gottschalk et al. [2008]. The authors suggest a method and language extension for making process models configurable in the sense that parts of a process model can be switched off/on or may be conditionally in-/excluded for a particular context of use. Depending on the modeling formalism simple on/off switching of activities may lead to violations of formal correctness criteria. The proposed approach implies that a reusable process model has to incorporate a rather large set of pre-modeled process variants which inevitably increases the model size. However, configurable process models that incorporate domain independent process knowledge which in turn can be adapted to a specific context are expected to reduce the effort of modeling. Complementary to the above approach van der Aalst et al. [2010] propose a Petri-net based formal foundation to ensure syntactical correctness of configurable process models that is largely independent of the notation used.

Müller et al. [2004] propose a system that enables the predictive or reactive adaption of workflows due to state changes of an object under work (e.g. a patient under medical treatment). Through a priori specification of rules that constitute potential logical failures (situation inadequacies) of a workflow and a specification of change operations that have to be performed to adapt the workflow to the new situation. Additionally, the approach suggested supports specification of certain constraints to limit the set of possible change operations (e.g. an additional step of treatment is added only when the medicine required for this step is available within a certain time frame). To estimate time constraints which are especially important in medical workflows both predictive (based on past cases) and reactive estimations of time constraints are applied.

Lee et al. [2007] suggest a conceptual framework to structure and specify typical issues (problems and opportunities) in scientific workflows along with a set of workflow adaption strategies for autonomous resolving of issues in case they occur.

Hallerbach et al. [2008] developed an approach where process models are adapted by applying only those options relevant in the given process context. Based on the values of context variables and a context model that relates these values to a set of predefined change operations a base process model is automatically adapted to fit the specific requirements of a new context.

Grambow et al. [2010] present an approach where software engineering process models are automatically adapted to the actual situation of a project utilizing situational method engineering as a method. Accordingly a basic workflow for each past case/situation is extended with activities matching a new situation. To be able to choose appropriate activities for a new artifact and situation, relevant activities are related to product and process properties of past situations or cases. These relations are expressed through so called selection rules that specify the conditions (process property values and combinations hereof) under which certain activities apply. The sequential order of activities is ensured through a set of simple semantic constraints – a predefined sequencing of the activities – that define binary relations for all activities so that every allowed sequences of activities can be determined. As the authors admit conditional flows, parallelization and other workflow patterns are currently not supported.
4.4 Process model modularization (A4)

Creation of reusable process models is the prerequisite of reuse. However, the primary objective of model creation is the transformation of an informal domain description into a new process model that can be used for a particular purpose. Making a process model a reusable asset is therefore a secondary objective that must not be followed in the first iteration of the model creation process.

(→R1A4) Modularizing a process model must be supported in two ways. First, a priori creation and specification of a reusable model must be supported. Second, a posteriori division of a large process model into smaller sub parts must be supported as well. The latter requirement stems from the fact that process model creation is practiced in an iterative manner where the final structure of a process model cannot be anticipated in advance.

(→R2A4) Division of a large process model into smaller sub parts should be supported in a way that allows for a user to easily recognize potential modules and offers as well a mechanism to split a large process model into smaller process models without corrupting the formal correctness (e.g. Küster et al., 2008). Small sized modules have been found to increase the likelihood of reuse (Holschke et al., 2009) and understandability (Reijers and Mendling, 2008).

(→R3A4) In either case the specification of interfaces for input/output and required resources must be possible, thus being able to integrate the module with other process models. Input/output specification may refer to physical resources (e.g. a particular material or item) or information resources (e.g. data objects). In some reuse scenarios it should be possible to assign a unique address to a module that makes it referable from different contexts without namespace conflicts (e.g. Ma and Leymann, 2009).

Related research  Regarding the decomposition of process models into fragments Küster et al. (2008) suggested an approach that is based on the idea that a process model can be decomposed into so called single-entry-and-single-exit (SESE) fragments where each SESE fragment is a non-empty subgraph in the process model with a single entry and a single exit edge. The fragment which surrounds the entire process model is as well considered a SESE fragment which is referred to as root fragment. So called canonical fragments are those fragments on the same hierarchical level which are not overlapping for a given process model. As a result a process model can be represented through a hierarchical structure of SESE fragments.

Ma and Leymann (2009) suggest a concept and XML extension to BPEL that allows for the definition of reusable process fragments. The basic idea is that the BPEL reuse concept which is limited to reuse of web services in BPEL process descriptions is extended to process fragments that – given that they are adequately specified – can be reused in other BPEL process descriptions.

In Thom et al. (2008) the authors present results of their research in recurring workflow activity patterns. They identified seven workflow activity patterns that they found to be sufficient to model a wide variety of processes. They propose a concept on how to make use of the presented activity patterns for reuse for the purpose of assisting users in the design of high-quality process models, e.g. by providing context-specific recommendations on which patterns to use.

In the latest BPMN specification so called linking elements (link events) are suggested to partition and connect process models horizontally. Additionally, a special element type (call activity) can be used to call/refer to other process models. For call activities and their respective callable elements data interfaces can be defined in a way that allows for a data flow modeling across arbitrary levels of detail. A more simplistic way to encapsulate part of a process into a reusable element is offered.

\footnote{http://www.bpmn.org}
by the sub-process element type. This element type allows to hide behavioral details of a process. However BPMN does not offer a construct to define such sub-processes to be reused in other parts of a process model.

Similarly ARIS Event-driven Process Chain Notation offers an element type that allows for the horizontal and vertical decomposition of a process. However, the semantics of this modeling construct is not well defined.

4.5 Process model annotation (A5)

Apart from the actual content of a process model contextual information regarding its creation, use and purpose are important for reuse evaluation [Fettke et al., 2006].

→R1. A modeler involved in model creation adds such information or at least links to such information in case it is available somewhere else (e.g. a document management system). As mentioned above (see →R2) generally three types of information can be provided: contextual data (e.g. the project where it was used), meta data (e.g. title, creation date, author) of a process model and the actual content which is manifested through the classes of model elements used and their concrete instantiations. This information may partly provided automatically through the modeling software (e.g. user data, time-stamp, modeling language) or must be able to be added manually.

→R2. In practice not all information contained in a process model needs to be studied to evaluate a process model for reuse. Therefore a modeler must be able to outline those parts of a model that are relevant for reuse evaluation (e.g. the input/output specification) whereas other details might be explicitly hidden.

→R3. A user should be supported in systematic and consistent labeling of process model elements as it improves not only understanding [Mendling et al., 2010] but also the reuse of process model modules. Although an a priori definition and limitation of allowed terms seems too restricting in terms of creativity it would be beneficial when a modeler is supported and encouraged in reusing terms common to a particular domain, e.g. a terminology or ontology (e.g. [Born et al., 2007]).

→R4. In addition to labeling of model elements a modeler may classify or tag process models to facilitate retrieval. A modeler must be able to adhere to some kind of predefined classification scheme for classifying process models. These classification schemes may be domain specific or domain independent. Predefined classification schemes are intentionally restrictive as they aim to make processes comparable and reusable regardless of a particular application context (e.g. Elias and Johannesson [2012]).

→R5. In contrast to standardized classification schemes a user should be as well be able to add personal categories, tags and keywords to a model that reflect his personal associations and way of structuring information. It should as well be possible to share such personal annotations with others and collaboratively develop annotation and classification systems (also referred to as “social tagging”).

→R6. A particular type of annotation are recommendations and ratings. A recommendation for reuse can be obtained by other human agents [Koschmider et al., 2010] or can be the result of some sort of computational logic, e.g. behavioral profiles [Weidlich et al., 2011].

→R7. Process models are created incrementally rather than in one step. Therefore process models often remain incomplete and incorrect for a certain period of time. A modeler must be able to assign information about the current state of the model. For reuse mainly mature and stable process models (already implemented models) are of interest. For reuse evaluation also the history
of a model’s evolution might be of interest (e.g. [Bae et al., 2007]).

**Related research** [Born et al., 2007] suggest a prototypical tool that supports a modeler in annotating process models based on an extended BPMN meta model and business domain ontologies. Using the tool a modeler is able to label and annotate model elements in a semantically precise and consistent way that ensures effective querying of large process model repositories and therefore facilitates reuse of process model elements. A similar approach is proposed by [Francescomarino et al., 2009] where business domain ontologies are used to give a precise semantics to the terms used to annotate business processes.

[Thomas and Fellmann, 2007] suggest a lower level semantics that builds on an ontology of business concepts and enables a modeler to select for example domain specific activities to build a process model and ensures consistent naming of activities throughout all process models. It is suffice to say that in practice such an ontology based language for model element labeling requires a rather large vocabulary to cover all possible concepts of a domain.

[Hinge et al., 2009] present an approach towards semantic effect descriptions of BPMN process models. Accordingly, a business analyst is able to describe the immediate effects of each task in a “practitioner-accessible controlled natural language”, which enables formal specification of effects using a limited repertoire of natural language sentence formats. These descriptions are then automatically to obtain cumulative effect annotations for each task in a process. By effects the authors refer to what an activity actually does, e.g. manipulating a business object, changing values of variables.

Regarding classification systems (taxonomies) for business processes several industry standards exist to date. For example, the Process Classification Framework (PCF) of American Productivity & Quality Center (APQC) aims at establishing a set of business processes practiced by most organizations, categorize them, and align them according to a hierarchical number system. A similar approach was taken within the context of the MIT Process Handbook Project [Malone et al., 2003] with the goal of providing a classification and systematization of business knowledge in terms of processes. An approach to standardize process vocabularies and process logic has been followed in the context of reference modeling (see for example [Fettke et al., 2006]) which resulted in many domain specific process frameworks like for example eTOM [3], ITIL [4] and SCOR [5]. These frameworks are primarily targeted in providing guidance in standardization, alignment and integration of business processes between organizations.

### 4.6 Process model storage (A6)

A process model to be reused must be retrievable. Therefore a process model artifact is usually stored in a repository. A modeler who has created a new process model or has adapted an existing process model stores the process models, the related data and indexes, classifications to enable fast querying, searching and navigation of the repository [Yan et al., 2012]. A repository might be implemented on top of a file system or database management system. In any case process models and associated data can be stored both in an internal format, for example in database tables or an external format that corresponds to the modeling notation or language used. For the modeler it is not of primary interest which internal format is used but however may be interested in fast and unified processing of process.

---

3http://www.tmforum.com
4http://www.itil-officialsite.com
5http://supply-chain.org/scor
models regardless their their external format. The external format is important when process models are exchanged with other software systems, e.g. an other modeling tool or a workflow management system.

(→R1.6) A process model must be at least created in a way that allows to read out and interpret the content along with the semantics of the concepts (model elements) included in the model. Several modeling language specific formats exist to date that support these requirements, e.g. BPMN XML, EPML, XPDL, BPEL.

(→R2.6) However, such formats are designed to describe the semantics of a business process on a very high level of abstraction, e.g. that the model element with label "confirm order" refers to an activity. Therefore a lower level semantics that describes and explains model elements used in a way that is oriented more towards the peculiarities of an application domain language are beneficial [Thomas and Fellmann, 2007], e.g. that "confirm order" is a special type of confirmation activity.

(→R3.6) Usually process models are created through diagrammatic (graph-based) representations. Although not necessarily needed to describe the structure and behavior of a process also graphical attributes like geometric positions of elements on a canvas, colors and size attributes must be stored along with the structural and behavioral attributes of a process model [Elias and Johannesson 2012].

(→R4.6) As stated above, process models are created in an incremental and iterative way. Therefore a modeler must be able to store incomplete, unvalidated and unverified process models to take up modeling at a later point in time. In the course of model creation a modeler sometimes decides to switch back to a prior version of a model. In other cases prior versions and variants of a particular model are used to keep track of changes and make comparisons [Elias and Johannesson, 2012; Dijkman et al., 2012].

Related research In [Elias and Johannesson, 2012] the authors have outlined several requirements for process model repositories. In their study they examined process model repositories in order to identify challenges that limit their usage in practice. In their evaluation of process model collections they found insufficient support for storing versions of models, annotations and contextual information. In Dijkman et al. [2012] the authors refer to various management techniques for efficient storage of process models and note that process models in addition to an internal format must as well be described through an external format that allows reuse in other contexts.

Review of selected process modeling environments

ARIS Business Architect 7.2.3 (ARIS)

Process model search  ARIS offers several features to perform a search within a model database. One such feature is the “Find..” feature. A user trying to find a model in ARIS needs to first specify a database the search should be performed in. A search across multiple databases is not possible. Next the type of object to be searched for needs to be specified. Three options are available that refer to the data model of ARIS which distinguishes between groups, models and objects. Having specified the type of object a modeler may search for a model with a known or partially known name. Furthermore it is possible to specify attributes and their values to search for. This is useful for example when all models of a specific author or all models with a certain state are searched for. However for reuse it would be as well beneficial if a user can specify a specific input/output variable which is not supported by default. All these attributes can be combined by logical expressions such
as AND, OR. A feature that is missing is full-text search in the sense that for a given search term all model related data is looked up. Rather a user has to decide whether to search within model names or within other text attributes on model level but is not able to search within the model elements and their attributes. This is only possible when searched for objects (model elements) but subsequently results in a list of model elements without having the information where the model element belongs to. In addition to the above described standard way of finding models also custom queries can be used. These queries created within the administration module which allows pre-configuration of principal input and output parameters to be used when executing a query.

The maximum freedom in formulating a query is offered through the “Scripts” module where a object-oriented scripting language allows to make use of the ARIS object model for programming all kinds of queries. These fully customizable queries can be stored and reused at any later point in time and allow both for the individualization of a search regarding input and output options. However, ARIS does not offer a way to specify requirements in a high level language (e.g. use-cases) and then subsequently retrieve matching process models.

Regarding the execution of the query and evaluation of candidate process models within the “Find” and “Queries” features it has to be noted that search results are neither assigned a score nor are they ranked with regard to their relevance. Rather search results seem to be returned in order of their occurrence or index in the database.

Process model inspection and selection  A search in ARIS returns an array of resulting models or objects. In the case of a standard search (“Find”) the results (if so) are returned as a table with columns for name, id, type of model/object. It is not possible to select the information (columns) needed for model inspection. Additionally no diagrammatic inspection and comparison of models is possible through the results table. This must be accomplished through selecting individual models from the results and subsequently open them or view them in a split screen editor where also differences are highlighted and classified. Through customized query scripts a ARIS user at least theoretically might overcome the above shortcomings as he has a maximum freedom in designing the way search results are presented to the user. Having selected a process model a user is able to copy a model (or model element) and paste it as a new model which can be adapted for reuse. Copying a model can be done by simply selecting a model and creating a copy of all contained model elements and therefore is independent of the original. Or one can create a copy that reuses the model elements from the original model by only referencing to these original elements which implies that changes to such an element will be propagated to all other instances of it. When copying a model a user might as well indicate that the new model is a variant of the original. In the latter case it is implicitly assumed that the variant model elements are independent from the original.

Process model adaption  Having selected a process model for adaption ARIS does provide help in evaluating the possible impact of changes through freely programmable evaluation scripts. In addition a user is able to inspect model dependencies and perform structural, syntactic and semantic checks. An adapted model can be stored and tested in a local database before it is transferred to the repository. Although transforming of models is partially supported, e.g. a EPC can be transformed into a BPMN model, it is not possible to make process models configurable through a higher level language that expresses the context of use (e.g. requirements). In other words, one cannot define parameters or high-level models like use-cases that trigger process model adaptions. Hence, according to the documentation transformations, e.g. between UML models and process models are possible to configure. As ARIS supports process model versioning and variant management adaptions to process
models are evident and can be traced back and compared.

**Process model modularization** In ARIS any process model can be potentially reused by other models through so called assignments or references. This means that an activity in one process model can refer to a whole process model implying that the referred process model is describing the behavior of this activity. Similarly links can be provided between process models on the same level. Regarding a posteriori division of a process model into smaller parts ARIS does not provide an explicit feature that guides a modeler in recognizing potentially reusable modules. Rather a modeler may cut and paste parts of a model into new model and then provide links. However, ARIS offers a feature to extract so called fragments from a process model. These fragments are meant to be reusable model patterns that can be stored in a separate library and can be easily reintegrated into any other model. In ARIS process models can be described by use of various languages (e.g. BPMN, BPEL, EPC). These languages support the specification of input/output behavior to a certain extent. For example, BPMN offers an explicit language construct to specify input and output data objects via a special kind of model element. Hence, ARIS does not provide a uniform and language independent way of providing such interface information. Any process model is stored with a unique id which makes it possible to reuse unambiguously within other models.

**Process model annotation** In ARIS there is no easy way of adding textual annotations. Hence, ARIS provides only free form text fields that can be placed anywhere in a process diagram. Another way of adding annotations is via descriptive attributes which may be defined in advance of a modeling project and subsequently used via the property window. Through this property dialog also metadata can be provided like the state or title of a model. Meta-data like time/date of last change, creator are provided automatically. Contextual information can be provided as well via dedicated attributes and via external document links on model group level or model level. Model groups serve as a folder structure to organize or classify models in a hierarchical structure. Interestingly the so called fragments which are reusable model patterns cannot be supplied with any meta-data, interface information or contextual information. Regarding labeling of model elements a use is only supported regarding the spelling of words. This allows only for a minimum grammatical consistency but does not foster consistency on a semantic or ontological level.

**Process model storage** Process models can be stored in databases and folders (or groups). Although ARIS has his own proprietary format to store models although an open ARIS specific XML format is offered to im-/export process models. BPMN models can be exported in BPMN XML. Transformations of models into some standardized format like BPEL, XPDL, BPMN are supported via programmable scripts. All models can be exported as a graphic to reused in other document types. ARIS can be set up to share models among multiple users via a central database that allows as well to configure access rights and versioning of models. All models are not required to be correct or complete before being stored in the database.

**ADONIS:CE 2.0**

**Process model search** Formulation of search question can be accomplished by either using predefined queries and their parameters or manually through a query editor. ADONIS features a proprietary query language (abbreviated AQL) that allows for a maximum freedom in building a search query. A major advantage of this query language is that it allows to explicitly refer to the semantics of model
elements, e.g. attributes, object (model element) classes and relation types. The predefined queries in ADONIS allow also searching within a process hierarchy. Thus, it is possible to specify a process model and retrieve models which are referring or are referred to directly or indirectly.

The specification of search space is performed implicitly through the ADONIS database. Searches performed in ADONIS are always performed for the whole database, there is no way to limit a search to a model group.

A search query in ADONIS return a list of search results that does not reveal any information about how relevant it is regarding the search question. Hence, the list of results returns the found process models in the order of their occurrence in the process hierarchy which adheres to an alphabetical order.

**Process model inspection and selection** Search results are presented in a extra window and table that is customizable regarding the model attributes and meta-data to be shown. However it is not possible to gain insight in the model itself and compare models regarding their internal behavior. For this purpose each model must be opened and inspected by switching between process model windows. Although properties of a model can be easily examined within the results list the external behavior of models regarding input and output variables can not be inspected without going into the model itself. This is due to the fact that external behavior is mainly dependent on the modeling language and meta-model used. ADONIS does not offer a feature to estimate adaption effort from a set of requirements given but allows to add comments to a model. Ratings or votes as a decision aid for a model are not supported. However, it is possible to inspect the frequency of references to a model. Once having decided for a model to be reused it can be saved as a new model and subsequently adapted or referenced.

**Process model adaption** When adapting a model to fit the requirements of a new context a model cannot be validated against a meta-model to see whether adaptions have violated any constraints. This is a general shortcoming of ADONIS. Hence, some structural checks (e.g. whether an activity has an incoming and outgoing sequence flow) are performed during modeling. Auto-adaption of a process model given a set of requirements is not possible. Versioning and maintaining links between model versions can be realized by extending model names with version numbers which does not support later reconstruction of model adaptions.

**Process model modularization** Process models in ADONIS can be attributed with various information but there is no standardized way to describe interfaces of a process with it's environment. This is mainly due to the fact that interfaces namely the input and output view of processes varies with the modeling language used. A posteriori partitioning of process models into smaller subparts is not guided or supported in a particular way. Rather one has to copy/paste parts of a model and then manually connect or refer to those parts.

**Process model annotation** ADONIS allows to add meta data, comments, description, current state and keywords. Several meta model specific attributes can be maintained. Unfortunately it is not possible to add manage keywords in a way that consistency of keywords is guaranteed. A classification system can be set up through model groups which can be organized into a hierarchical structure. Model can be linked to external documents and web resources. Annotation of process model parts can be accomplished by using so called notes which can be linked to model elements. As mentioned above models can be assigned version numbers and a record of changes can be maintained.
although not automatic. Rather changes need to be documented manually in a so called notebook. ADONIS does not allow to add quantitative ratings or recommendations to a model. It is possible to obtain statistics about the number of references to a model. Consistent labeling of process models and related elements through a built-in ontology or thesaurus is not supported.

**Process model storage**  Process models are stored in a database. Apart from a built-in database engine also several others are supported (e.g. Oracle, MS SQL Server). It is possible to export process models in a reusable format like BPEL, XPDL, ADONIS XML, ADONIS ADL but it is not possible to import standard XML process formats. It is possible to store incomplete and incorrect versions of a model. A version management feature to trace revisions and variants of a model is missing.

**iGrafx Process 2011**

**Process model search**  In iGrafx model content is managed through documents which may contain multiple diagrams of all sorts. Through an optional component “Process Central” a database can be used to share content among multiple users. iGrafx enables a modeler to search for database items. Database items can be documents or components (diagrams included in document). It is possible to use full text search along with wildcard expressions for searches within the database. Additionally classes of items may be included or excluded (e.g. files). Through the “Query Builder” feature attribute types and operators can be used to formalize a query. However, there is no process specific query language but a largely language independent data model. Queries can be executed for one repository at a time but not across repositories.

**Process model inspection and selection**  Query results can be customized regarding the fields (attributes) of the models to be shown in the list. Resulting models can be selected and subsequently edited but there is no way to compare a selection of models. The frequency of model reuse is not indicated but it is possible to see for each model element where it is currently used. Although properties of a model can be easily compared within the results list the external behavior of models regarding input and output variables can not be inspected as there is no language independent way to specify input/output parameters. iGrafx like the other tools examined does not offer a feature to estimate adaption effort from a set of requirements given but allows to add comments to a model. Ratings or votes as a decision aid for a model are not supported. A model chosen can be saved as a new model and subsequently changed.

**Process model adaption**  Adaptions to a model can be validated against rules. Impact analysis that takes in/out references into consideration is not supported. Similarly automatic adaption based on high-level requirements are not supported. It is possible to use parts of a model (a diagram) in a new document which makes it possible to change/add only those parts that need adaption.

**Process model modularization**  iGrafx does not support a modeler in partitioning a model into reusable parts. However through a flexible data model that separates diagrams from documents a diagram can be "shared" across documents through links. This approach makes it possible to easily outsource certain parts of a model in a later phase of modeling. There is no general way to specify input/output behavior on process level but on activity level. There is also no way to store fully specified reusable activities and process model parts in a library.
Process model annotation  In iGrafx arbitrary repository folders can be created and nested within each other which is a way of classifying process models hierarchically. iGrafx automatically maintains several meta-data attributes like date of creation, last modification, user name, state. It offers as well features to link models and model elements with other iGrafx documents but also with resources from the web or local directory. Personal annotations are supported via the optional module “Process Central”. Also recommendations and votes can be added. Consistent labeling through some kind of thesaurus or ontological support is not possible.

Process model storage  Process model documents are stored in a database via an optional component but can be stored as well in a local folder. Via an add-on component standardized formats for reuse (e.g. BPMN, BPEL, XPDL) can be produced but not re-imported. Diagrammatic data is stored with the pure logical data as well and can be exported in various formats, e.g. png, bmp, svg. Import of process model data from other applications (e.g. Visio data) is possible but not from standard formats like BPMN, BPEL. It is possible to store incomplete and incorrect versions of a model and maintain information about the current state of a model. A version management is possible via “Process Central” where revisions of a document are tracked and can as well be restored.

5 Summary and Conclusion

The above described research is targeted at providing a process-oriented perspective on process model reuse. Main activities involved in process model reuse have been systematically identified from a comprehensive review of relevant literature. Subsequently, these activities have been synthesized into a process model of process model reuse. This generic model can be can be considered a structured set of meta-requirements that need to be fulfilled by process modeling environments in order to effectively support process model reuse.

The major contribution of this work is that it summarizes prior studies in the field of process model reuse and connects as well to concepts from related research areas such as software reuse and knowledge reuse. The process model developed along with associated requirements is intended to serve practitioners and researchers as a frame of reference and starting point when designing and evaluating reuse-oriented process modeling environments.

To show the applicability of the evaluation framework it has been applied to three state-of-the-art process modeling environments. The findings of this qualitative evaluation show that all three modeling environments support reuse only for selected activities. Support for the whole process of reuse, that is for “model-by-reuse” and “model-for-reuse” activities, could not be found to be implemented sufficiently. In particular, support for “packaging” reusable process model components, and their effective retrieval lacks state-of-the-art feature such as annotations, recommendation-based retrieval and integration into new process models.

To prove the applicability and validity of the conceptual model of process models reuse in practice further evaluation studies need to be performed, especially with the involvement of practitioners and additional modeling environments. Potential limitations that we are aware of is that the framework does not account for non-functional requirements of modeling environments. However the framework is intended to serve as a starting point for requirements elicitation in the course of developing process model environments and as well is a low-cost way of heuristic evaluation, e.g through walk-through or scenario based evaluations.
References


