Association for Information Systems
AIS Electronic Library (AISeL)

Wirtschaftsinformatik Proceedings 2011 | Wirtschaftsinformatik

2011

Similarity Determination in Activity Sequences – A Supportive Framework

Jörg Schmidl
Technische Universität München, schmidl@in.tum.de

Holger Wittges
Technische Universität München, wittges@in.tum.de

Helmut Krcmar
Technische Universität München, krcmar@in.tum.de

Similarity Determination in Activity Sequences – A Supportive Framework

Jörg Schmidl
Technische Universität München
Boltzmannstraße 3
85748 Garching, Germany
+49 89 322 99 83 10
schmidl@in.tum.de

Holger Wittges
Technische Universität München
Boltzmannstraße 3
85748 Garching, Germany
+49 89 322 99 83 13
wittges@in.tum.de

Helmut Krcmar
Technische Universität München
Boltzmannstraße 3
85748 Garching, Germany
+49 89 289 19 530
krcmar@in.tum.de

ABSTRACT

An increasing number of information systems support their users by helping them in reusing existing knowledge and experience. Often this is done by retrieving similar instances like similar documents, similar process executions or similar persons. While the recommendations use similarity as central concept, the selection of a suitable measure is often done by intuition. This paper introduces a framework that supports the application engineer in selecting and configuring a suitable similarity measure. The requirements of the intended framework are gathered before the architectural implications are detailed. The resulting framework is applied in a case study in which project performance prediction is to be supported by the similarity of the projects' activity sequences. The results show the framework's utility by allowing a comparably simple configuration to yield a considerable support in selecting and configuring a suitable similarity measure.

Keywords

Similarity Measure, Activity Sequence, Similarity Framework, Business Processes

1. INTRODUCTION

Today, support by Information Systems (IS) is omnipresent in organizations. In particular the trend of increasing knowledge intensity of everyday tasks leads to the increased use of IS to support the knowledge worker [7]. IS typically log users' activities for administrative reasons e.g. authorization control or for analytical purposes e.g. web server logs to determine usage behavior. Activity logs are also increasingly used by applications for the support of the end user. These applications provide recommendations based on the activity logs. For example, they deliver similar items to the user depending on his previous searches for other items [19]. Other software recommends the next steps in a process, based on what has been done so far and what others have done in a similar situation [23; 25; 21]. Yet other support systems recommend who to contact in a social network based on the contextual overlap of the two [16].

The common theme among these support systems is their reliance on the concept of similarity to support the user. They recommend similar documents, similar process steps or similar persons, contingent on the current context, which needs to be modeled in an appropriate way.

While all these applications build on the concept of similarity, the interpretation of why two or more objects are to be considered similar depends on the application and its use cases for which the similarity-based application is employed.

The challenge when using similarity-based applications lies in determining a suitable notion of similarity. This is a complex task. There are many approaches stemming from diverse disciplines. They build upon definitions of similarity that are specific to those disciplines. In addition, most similarity measures use a number of parameters to determine how the similarity between two objects is determined. Finding a suitable configuration of a similarity measure is, therefore, a challenge.

Responding to this challenge, this paper introduces a framework that supports the developer of similarity-based applications when faced with the selection and configuration of suitable similarity measures. The framework allows to configure different similarity measures and in a next step to evaluate their appropriateness for the target application. Features of various applications that use similarity determination are taken into account to determine requirements for the framework. The implementation based on these requirements is detailed subsequently. In a case study, we demonstrate the utility of the framework in creating a similarity-based application.

The paper is organized as follows. Section 2 gives an overview of related work. Section 3 derives requirements for a framework to support the selection and configuration of similarity measures for activity sequences. Section 4 presents the paper's core contribution, outlining the framework that meets the requirements as outlined in section 3. Section 5 presents a case study that shows the utility of the framework in a real world setting. The paper concludes with a discussion and recommends further research.

10th International Conference on Wirtschaftsinformatik,
16th - 18th February 2011, Zurich, Switzerland

2. RELATED WORK

This contribution introduces a generic framework that supports the application engineer in selecting and configuring a suitable similarity measure for activity sequences. Its benefits are twofold. Practitioners can use the framework with minimal effort to find suitable similarity measures for their applications. On the other hand it provides a basis for researchers investigating properties of similarity measures for activity sequences.

A similar approach was taken in the process mining discipline where the generic ProM framework supports the development and application of process mining algorithms [6; 30]. The acceptance of this generic framework inspired us to create a similar framework to support further research of similarity measures for activity sequences and their application. In analogy to the framework described in this contribution, ProM acts as incubator for new algorithms and concepts in process mining. Additionally, ProM also operates on activity sequences. However, ProM's central concern is the support of process mining, i.e. the recovery of process models from event logs. Although it has incorporated many additional concepts and algorithms that extend its use to more than only process recovery, the determination of a suitable similarity measure is not in the scope of process mining. This is why our framework makes use of ProM's facilities wherever appropriate for example when accessing data sources or creating process models if this is necessary for the similarity measure. However, our framework's core functions are too different to integrate them into the existing ProM framework as a plug-in

Some of ProM's plug-ins make use of the concept of similarity for example to predict execution times of process instances based on previously executed process instances [28]. This is similar to what we discuss in the case study in chapter 5. However, while our case study aims at finding a well-suited similarity measure underlining the utility of our framework, in [28] the authors only use one kind of similarity measure and describe how to adapt its parameters best.

The challenge to find a suitable similarity measure has also been addressed in another context. In [10] the authors investigated how similar users or similar content can be determined in different social media to increase its usage. In all their experiments the authors used the same similarity measure, but used nine different sources for similarity information investigating their effects on six different social media applications. The results showed that the source of similarity information had significant influence on the perceived quality of the system's suggestions and also that the influence varies with respect to the different applications. In this paper we also stress that similarity measures must be tailored to the application that makes use of them. Additionally, in our framework we also acknowledge the great influence of the initial data and in what format it is collected. However, while in [10] quantifying the influence of different data sources for a concrete application was the goal, we focus on creating a generic framework that could support tasks like the one in [10]. Similarly, in [24] the authors investigate the influence of different similarity measures on recommendations in an online social network. They applied six different similarity measures to recommend potentially interesting sub-communities to their users and investigate the influence of the measure on the quality of the recommendation. In contrast to the contribution at hand, the authors in [24], however, focus on one use case and not on a generic support tool. Also they only use similarity measures that operate on sets, while our framework allows the usage of other similarity features as well.

3. REQUIREMENTS ANALYSIS FOR THE FRAMEWORK

Here, the requirements for a framework to support the selection and configuration of measures for similarity-based applications are documented. Requirements engineering can be done in many different ways [13]. Sources of requirements can be for example domain knowledge, existing systems, users, standards or regulations. In the following we use existing frameworks, source systems, data format standards and applications that use similarity measures as source for requirements elicitation. Each requirement is annotated with a number for reference in the outline of the framework's architecture.

A similarity measure is appropriate if it supports the goals of the target application. Therefore the determination of an appropriate similarity measure consists of two phases: firstly, selecting and configuring a similarity measure and secondly, checking the fitness for the target application. However a prerequisite is having data that is suitable to act as source for similarity information. This step is particularly important, because the selection itself greatly influences the result of the similarity determination [10]. Therefore, a generic framework must have the three components as shown in Figure 1. In the following each identified requirement references its corresponding component as indicated in this figure.

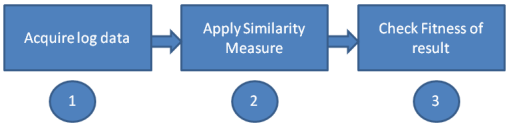

Figure 1 : Steps for determining suitable similarity measure

The requirements for the first component are elicited by inspecting different process aware information systems (PAIS) [8], such as ERP systems, project management systems and personnel management systems, the format of their data and how the log information is interpreted by the application and by persons.

Whenever it is desirable for an application to utilize the similarity of activity sequences, the first step is to retrieve those activity sequences from source application logs. But many applications may serve as sources for information and their data formats also take many forms. Some applications, for example ERP systems, store log information in databases while others such as web servers use files for this purpose, which are accessed through differently. A requirement for the framework, therefore, is:

> The framework should support both – data stored in files and in databases (*R.1a*)

In addition, the format of the data may vary. While data in a database is structured by definition, file-based logs can be stored

in comma separated value (CSV) files, using plain text with or without providing header information about the meaning of each section in a line, which is a still quite common solution. Other file-based logs are stored using some XML dialect with or without providing a schema definition along with it, especially when interoperability is important. Yet other IS use a log structure that is not plain text, is proprietary and needs parts of the IS logic to decode the log. Since all of these data formats are found in applications that can act as data source, the framework should fulfill the following requirement:

> The framework should support the retrieval of data from sources that are structured, semi-structured or use structures in a proprietary format (*R 1b*).

Additionally, the granularity of the log information may differ. Some applications log every user interaction, such as web servers, while others only log certain events including, for example, the change of a status indicator in a project management system. The same is true for context information that goes along with the log entry, which also can differ significantly in its extent. For example, while browsing in an intranet, much context information of an user is typically at hand, while anonymous access to an internet site offers less context information. Therefore, another requirement for the framework is as follows:

> The framework should be flexible enough to handle both rich data sources and to extract or amend less rich data sources (*R.1c*).

IS also differ in their pervasiveness. Some IS log user interaction in the background with little or no user involvement, such as web browsers while they browse through a website, while other IS only write into their logs when explicitly requested by the user, such as for example accounting systems. This influences the granularity and the possibilities for interpreting the log, because in the first case we often need to interpret implicit behavior while in the second case the intention of an user is more explicit and related more strongly to the log entry. Depending on the knowledge about the process that is supported by the IS, it is possible to amend log data with context information. This creates another requirement for the framework:

> The framework needs to be agnostic to how the data is captured from a technical point of view, but needs to provide means for amending the data with implicit information (*R.1d*).

Also, the kinds of stored data differ. Some applications store an event, or activity respectively, in their logs, i.e. what has happened. Others store data that reflects the situation after an activity has been performed, i.e. the result of what has happened. A web browser for example might store the event "page index.html has been requested", while a project management system might store the status "project budget is (now) 100k €", but not the event itself that increased the budget to this amount. Another requirement for the framework, therefore, is as follows:

> The framework should have the capability to transform log information containing status snapshots into log information containing status changes (*R.1e*).

Independent from the characteristics of the data source itself, more than one application log may contain information for an activity sequences, i.e. the information contained in one application log can augment information from another log. For example, a project management system could contain the execution history of a project, while in a separate accounting system, information about consumed budget is kept. This is why the framework should fulfill another requirement:

> The framework should allow the flexible and iterative enrichment of log data from multiple sources (*R.1f*).

Being able to import data from arbitrary data sources and being able to transform them in a suitable format forms the basis for the second component of the framework in Figure 1. It allows to apply similarity measures to the input data. The requirements for the second component were elicited by reviewing the properties of thirteen similarity measures found in literature, extracting their common features and deriving requirements from their common features. The measures were used in a wide range of disciplines such as protein function prediction in biology, comparison of Web Service definitions in computer science and overlap calculation in graph theory, to name a few application scenarios.

Before discussing different kinds of similarity measures, it is necessary to take the goal of their use into account. Applications that make use of the similarity of activity sequences can have different target functions. For example, in project controlling it is often relevant to assess the likeliness of success. This could be done by determining similar projects that have been completed already, taking their success as an indicator for the currently running project. In that case, the goal is to make a good estimation about project success. Another example with a different target function can be found in product recommendation engines where users are presented with similar products that overlap with their peer's preferences. In this case, the goal is to leverage the cross-selling potential. Different target functions have different definitions of when a similarity measure works well on a set of activity sequences and when it does not. It is often appropriate to adapt a similarity measure to suit its intended support for a goal, using supervised learning techniques [33]. The framework's similarity measure component should, therefore, fulfill the following requirement:

> The framework should have the capability to label a training set of activity sequences with an indicator of its utility in relation to the target application's goal (*R.2a*).

There are a number of different ways to determine the similarity between two entities. For this reason the framework needs to be flexible enough to support each different way. In a first instance, an entity can be described by certain flat attributes, for example a project is described by the number of project members and the total budget. In that case, the two entities can be compared according to their attribute values, where the comparison can be done with different algorithms depending on for example the data types or data ranges. This is why the framework should fulfill the following requirement:

> The framework should be able to support similarity measures that operate on input entities that are described by attribute-value pairs (*R 2b*).

In a second instance, an entity can have structured components, for example a project is described by the activities that have been performed during its execution. In that case, the two entities can be compared according to the overlap of the same constituting parts, i.e. the same activities. The framework should, therefore, fulfill another requirement:

> The framework should be able to support similarity measures that operate on structured input entities by for example comparing the overlap of components (*R.2c*).

Additionally, the constituting parts can themselves have attributes, for example each activity in a project can have a specific person that is responsible. Therefore, the comparison of entities can be based on constituting components, acknowledging the difference in attributes as well. Essentially, this is an extension of the requirement described before, where the constituting components, were treated as flat structures and were compared for equality. The framework should fulfill the derived requirement:

> The framework should be able to support similarity measures that operate on structured input entities where each structured component is (additionally) described by attribute-value pairs (*R.2d*).

Finally, it is possible to take the relationship between the constituting parts into account. The relationship represents the temporal or logical order of the constituting parts and may also reflect interleaving of those activities. Addressing this fact, the framework should fulfill the following requirement:

> The framework should be able to support similarity measures that operate on the structure of its entities, i.e. that use structural properties of the input data for similarity determination (*R2.e).*

Additionally, many similarity measures use one or more parameters to configure the computation of similarity. The framework must, therefore, fulfill an additional requirement:

> The framework should offer the capacity to process parameters for each similarity measure that determine its behavior (*R.2f*).

Each similarity measure typically focuses on one or at least a small set of properties of the input object. However, it is possible that the desired notion of similarity is best reflected by a combination of different properties. In this case, the simultaneous application of different similarity measures is necessary. This poses another requirement to the framework:

> The framework should allow for a compounded calculation of similarity using different measures (*R.2g*).

As indicated above, the structural properties of activity sequences can be used for the similarity determination. Yet, each activity sequence itself has a linear structure by definition. To find out about the dependencies between activities, a model of possible sequences indicating their relationship is required. In many cases, explicit models of activity sequences are not available because they are too expensive to create or because the activity sequences are too flexible to render a model useful. Nevertheless, if the usage of structural properties is deemed necessary, there needs to be a way to at least recover an implicit model for the activity sequences. It would have to be reconstructed from the IS logs and would then indicate the process "as it is lived". In terms of similarity determination, it can be used to deduce structural properties of an otherwise linear activity sequence. The framework should fulfill another requirement:

> The framework should provide a possibility to create a (process) model using the activity sequences that are available (*R.2h*).

We have reasoned above, that similarity measures only have a purpose with respect to their target application. Therefore, our framework should facilitate the selection and configuration of an appropriate measure. To find out about the utility of the selected measure and its configuration, the results have to be seen in the light of the application that they will be used for. The framework needs to be integrated into the target implementation or needs to be integrated into a suitable representation thereof to show its utility. Especially if supervised learning techniques are used for the selection and configuration of similarity measures, the feedback of the application about how well-suited the similarity measure's results are for the intended use is pivotal. Therefore, the framework needs to fulfill a requirement that intentionally covers a broad spectrum of interpretations to encompass arbitrary similarity-based applications:

> The framework should support the integration of similarity-based applications or suitable representations thereof that consume the results of the similarity determination and give the framework feedback about the quality of the results (*R.3*).

4. PROPOSED ARCHITECTURE

The requirements discussed above informed the design of the framework. The logical structuring in three different components (see Figure 1) proved useful for the elicitation of requirements. For the implementation of the necessary functionality it turned out that the second component can be split into three modules: one that supports classifications for supervised learning, one that can mine a process model from input logs for the support of structural similarity measures and one for the application of similarity measures itself. Therefore the architecture features a modular design with five main modules (see Figure 2). The framework handles the flow of action by instantiating one or more plug-ins for each module and passing on the control subsequently. However, it is not mandatory to use all modules, i.e. classification of activity sequences is only necessary when supervised learning should be supported and the creation of an activity sequences' model is only necessary if structural properties should be used in the similarity determination.

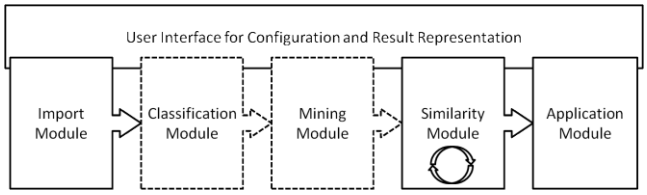

Figure 2 : The framework's modules

The framework was implemented using Java, because it is a very common programming language that many application engineers can work with. Wherever appropriate, existing applications and frameworks were integrated into the framework directly. This is true for parts of the import module that builds upon the ProMImport application and the process mining module ProM, that relieved us from the effort of implementing process mining algorithms. If the functionality of the incorporated applications did not entirely suit our needs, we extended them to meet our set of requirements. In this section, the framework's five modules are detailed. The reasons for the design choices in each module are explained by reference to the requirements in chapter 3.

4.1 Importing Data into the Framework

The requirements analysis showed that activity sequences are frequently stored in different locations, i.e. databases or files, and have different formats. Therefore, importing data into the framework for similarity determination cannot be done with one single import routine but rather must be done by offering an interface that supports as much flexibility as possible. A plug-in that implements this interface then offers the application-specific ability to extract activity sequences.

Extracting information from logs is a common challenge. An existing tool, the ProMImport Framework [9], served as a good basis for the import module, although its extraction mechanisms primarily focus on the support of process mining. For its use in the framework, ProMImport had to be extended. The ability to store the extracted data before displaying it and the ability to connect the output of one import plug-in to the next import plug-in were added. In this way, requirements *R.1a* and *R.1b* are addressed by delegating the specifics of the data extraction to plug-ins while offering a generic interface.

The requirements analysis also shows that relevant information for activity sequences could be spread across different logs that logically complement the information contained in each. The ProMImport Framework had no support for aggregating information found in different kinds of sources for one combined import result. Therefore, the concept of chaining importer plug-ins has been implemented into the framework. While ProMImport would display the results of the import directly, the framework's import module can deliver the results to another importer plug-in to augment the results, yielding, in the end, one integrated import result. This fulfills requirement *R.1f*, which states that information could be spread across different sources.

Offering the ability to connect arbitrary data sources also requires that a suitable data format is defined within the framework that can be used for intermediate storage. As the requirement analysis indicated, logs can have varying expressiveness concerning the granularity of logged activities and concerning each activity's context information. The internal data format must be designed in such a way that the transformation of source data into this internal format is without loss of expressiveness or limits the loss to a minimum. Data formats that are crafted like this can be found in the WFMC's specifications of the Common Workflow Audit Data (CWAD) [32] and in the MXML format [29]. Both data formats are abstract enough to represent the contents of different application logs. Being tailored to the needs of a workflow system, the CWAD format has a considerable number of attributes that only apply in a workflow context. The MXML format on the other hand, abstracts from workflows and therefore allows a more straightforward transformation of arbitrary data. Therefore, MXML is adopted as the internal data format for the framework because it is flexible enough to handle logs with varying granularity and is abstract enough to handle logs with varying contextual data. This fulfills requirements *R.1c* and *R.1d*, which state that the granularity of data can vary.

The chosen data format conceptually stores sequences of events, which is also true for the CWAD format and many others. On the other hand the results of the requirements analysis pointed out that there are also logs that do not store activities as events but rather by storing the results of the activities. Consequently, there needs to be a mechanism to transform logs containing data states into event sequences. While this problem has been addressed in theoretical computer sciences [15], there is no actionable implication for an implementation as would be necessary for the framework. For that reason, the framework incorporates a configurable, XML-based application for this purpose. After one-time configuration, it automatically selects defined portions of the source data that contains sequences of states and transforms them into event sequences during data import. The events are created using activity names that are configured before importing the data. This fulfills requirement *R.1e*. In addition, for each activity attributes can be assigned and their values can be calculated using basic arithmetic and string operations. This fulfills requirement *R.1f*.

4.2 Classifying Activity Sequences

The requirement analysis specifies that a label must be assigned indicating the utility with respect to the desired target, if supervised learning is to be used. In principle, two different ways to allow for labeling are possible: automated or manual. In one case, activity sequences are labeled according to one or more rules that are created by a domain specialist. For example, the result of project executions as good, mediocre or bad could be automatically determined using the budget-to-spending ratio of each project as basis for a rule. However, this automated approach has a significant disadvantage. If there was a rule available that perfectly labels this type of activity sequences, then this rule is at the same time a perfectly suitable similarity measure and there would not be any need for using the framework in the first place. However, the more complex activity sequences are, the less likely it is that a person knows according to which measures an activity sequence should be evaluated. This person nevertheless may often be able to indicate the result tacitly without knowing how to derive this judgment formally. Therefore, the framework uses the other choice, namely the manual labeling approach. In this case, a person classifies a training set of the activity sequences and stores the results in a

csv file. This file is then used as an input source for the framework to automatically classify the training set of activity sequence. As outlined in the requirements analysis, labeling is not always necessary and is implemented as an optional step in the framework. Having the labeling module fulfills requirement *R.2a*.

4.3 Generating a Model of the Activity Sequences

The requirements analysis shows that many applications do not have an explicit model for the execution of activity sequences, but its users may follow implicit models that for example stem from corporate rules or from the technically possible interaction via a graphical user interface. If the implicit model can be explicated in a possibly only approximated model, it can be used to extract structural properties of an otherwise linear activity sequence. Inductively creating a model from instances contained in a log is the main concern of the process mining discipline [1; 3]. There are many algorithms available to mine a model from instances. The application of these algorithms is facilitated by the ProM framework [6; 30] that has many of them integrated as plug-ins already. However, ProM returns its mined models typically as event-process-chains (EPC) or as petri-nets [18] that perfectly serve the purpose of modeling the execution semantics. In the context of determining similarity, the execution semantics do not play a large role though, which allows for the simplification of petri-nets and EPCs into simple graphs that only consist of edges and vertices. This also creates the ability to use similarity measures that work with simple graphs.

The transformation is done as follows (see Figure 3). Whenever the ProM framework returns a petri-net[1], the framework needs to transform it by creating one node for each of the petri-net's transitions (blocks in the figure). Those nodes are connected to one another by inspecting which transitions are connected in the petri-net, where the term connected is interpreted as follows. Two transitions are connected if there is exactly one place (circles in the figure) in between them. If the model has explicit routing nodes (not shown in the figure; would be *XOR*, *OR* or *AND*, with the obvious semantics), then two transitions are connected if there is a sequence of zero or more routing nodes and one place in between them, but no other transition. Additionally the resulting graph is extended with explicit *Start* and *End* nodes that are implicit in petri-nets. Using this definition and this way of transforming the petri-net, it does not matter if one uses explicit routing nodes or implicit routing by means of petri-net firing semantics. Both will be transformed into the same graph. An example of implicit routing can be found in the petri-net on the left part of Figure 3. In that example Task 3 must always be executed in accordance to petri-net firing semantics.

[1] In the context of process mining and also similarity determination EPCs can be transformed into petri-nets without loss of relevant information

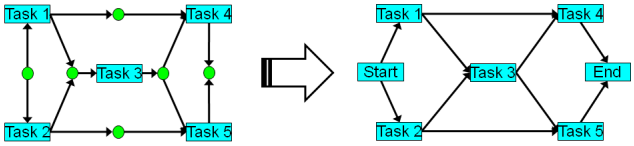

Figure 3 : Transformation of a Petri-net to a graph

The transformation is adequate. The argument is as follows. Since essentially places are removed from the petri-net, along with the routing nodes, if there are any, only the connection between transitions are left that are now considered nodes. It is, however, not obvious anymore if those connections are to represent exclusive execution or parallel execution. It is no longer known which transitions are prerequisites of certain nodes and, therefore, explicit information about the execution semantics is lost. However, this is not a problem for the intended use. The transformed model is solely used to identify the path that was taken by a process instance so far. Therefore, there is no need to know which connections previously modeled exclusive or parallel execution. The connections that are used by the instance were obviously parallel, the ones that have not been used but could have been according to the graph are of no interest, but obviously have represented OR/XOR split alternatives. It is also not a problem that there is no information about which nodes are prerequisites of another node, since the further proceeding of the process instance is of no interest. Utilizing the existing ProM framework but adapting its output fulfills requirement *R.2h*.

4.4 Determining the Similarity between Activity Sequences

The requirements analysis indicates that the framework must offer an interface for the creation of its own similarity measures. Nevertheless, it is desirable to have a reasonable number of algorithms available in the framework to make it useful from the start. As the algorithms differ with respect to the parts of the activity sequence they use for the computation, the framework should offer at least one algorithm for each kind of similarity measure. This guarantees that other similarity measures that operate on the same kind of input data can be integrated into the framework. The framework has 13 similarity measures integrated in its initial phase, that can work with the general properties (requirement *R.2b*) of an activity sequence, the overlap of activities (requirement *R.2c*), taking into account the activities attributes if necessary (requirement *R.2d*) and also taking into account structural properties (requirement *R.2e*).

Within the framework, the main task of each similarity measure is to determine a similarity matrix, i.e. it must create a matrix with as many rows and columns as there are activity sequences with each entry containing the degree of similarity between the respective combinations of activity sequences. This very general representation of a similarity measure's result allows applications to extract the relevant information flexibly. This fulfills requirement 5 which states that applications building upon the similarity of activity sequences can have diverse needs with respect to the similarity measure. A downside is, that this way of storing a similarity measure's result is not the most

efficient way considering computational cost and space requirements.

The requirements analysis also reveals that similarity measures frequently need configuration to a certain degree. The framework provides the means to properly configure a similarity measure. Each similarity measure plug-in is requested to publish its necessary parameters to the GUI component and can then process them as needed. This fulfills requirement *R.2f*.

Further, the requirement analysis shows that cases must be supported in which one similarity measure is not enough. A combined result of different similarity measures might fit the application's needs better than a single measure could. The framework supports this kind of configuration. Internally, each measure computes one similarity matrix. The matrices are then combined to yield one similarity matrix. The combination is done by also allowing for a weighting between the measures. This fulfills requirement *R.2g*.

4.5 Applying the Selected Similarity Measure in the Target Application

The framework is designed to support any application that builds upon the use of activity sequence similarity. This creates the need to offer different configuration mechanisms. One way of configuring the similarity application is the use of incrementable parameters. The parameters are set up with a maximum, a minimum and an initial value, along with a step size. The application iteratively performs its task and changes the incrementable parameter as indicated by the step size, until the upper or lower limit is reached. This functionality is indicated by the circle below the similarity component in Figure 2. The usage of the parameter is not controlled by the framework itself, but by the application, while the framework performs the increment. If more than one incrementable parameter is set for the application, the framework ensures that every possible combination is explored.

Additionally, the framework offers the application an interface for interacting with the intermediate results[2]. The concept is as follows: Directly after the intermediate information is created, the application is asked to pre-process the intermediate information. This happens before the determination of weight distribution and before using incrementable parameters. If the application uses this option, it can tell the framework how to pre-process the dataset and by this means adapt it to its needs. It is then provided with the pre-processed data instead of the plain intermediate results. This step is performed as long as the application indicates that it still wants to change the data. This implementation is generic enough to support arbitrary applications but offers enough functionality to still support the application engineer which fulfills requirement *R.3*. An illustration of the utility of this feature is part of the case study in chapter 5.

[2] The imported activity sequences, the potentially mined model, and the potentially created classification are considered as intermediate results.

4.6 General Features

The goal of the framework is not only to relieve the application architect of the task of finding a suitable similarity measure, but also to find a well-suited configuration of the similarity measure. The similarity application module supports this feature. When the usage of more than one measure is desired, the framework can be used to determine the best combination in terms of weighting. The user only needs to specify how fine-grained the search of the best solution should be by providing an increment value. This value is then used to exhaustively search the result space, which is done by iteratively using each weighting combination for the similarity measures. The combined measures' result is determined in the light of the application that builds upon them, which in turn informs the framework how well this combination is suited to its needs.

After each possible iteration that might stem from the presence of incrementable parameters application pre-processing calls or optimal weight determination, the application returns its collected information to the framework. The collected information reflects the respective performance of each possible combination. For this purpose, it uses a multi-dimensional array, where each dimension represents one incrementable parameter, and the array's value represents the parameterization's performance with respect to the application's performance criteria.

To enable the user to visually explore the relationships, the user can select a graph that shows a two-dimensional projection of the resulting multi-dimensional array. The two dimensions of the graph can be determined without limitation.

5. CASE STUDY

The framework's capabilities are investigated in a real life scenario, where the similarity of activity sequences is used to amend the functionality of an existing application. As one instantiation of an IS that benefits from similarity determination, in the case study, a project management system is investigated. The results of the case study inform the company how to make better use of what has been learned in previous projects with only minimal effort. On the other hand the case study shows the framework's ability to scale and support the application engineer.

5.1 Case Study Background

The company in our case study had been using a proprietary project management application that kept track of the status and the customer interaction during project execution for a number of years prior to the case study. It distinguished between nine different statuses a project can have, such as *customer contacted*, *price negotiated*. Additionally in each status, information like assigned employees, estimated project cost and profit and realized cost and profit are stored. Also an SAP system was used for keeping track of the employee's time on different projects.

The system contained a history of 124 projects covering consultancy and prototypical development of applications for customers. Each dataset contained predefined steps that indicated the status of the project, interactions with the customer, the

respective dates of these interactions and information about who is involved in the respective phases of the project.

Up until the case study, the project management system was used for (retrospective) project reporting and for giving the project portfolio manager an up-to-date overview of the status of the projects. There was, however, no actionable support feature, like project progress projection or reusing experiences from previous projects. The project members saw it as a valuable approach to find similar projects using the activity sequences that could be extracted from the logs. The assumption was, that helping a project manager of a currently running project in finding similar projects, would allow him to learn from the experience of similar previously finished projects. This way he would be presented with the likely performance of his current project utilizing the performance of the similar projects as a predictor and contact details of the related projects' managers to ask them for support.

However, the team members could not clearly define "similar" in this context. The project portfolio manager was able to indicate the quality of the projects' processes, which ranged from poor to good, without being able to state which parameters could be used to support his judgment. This is a typical problem in complex decision environments.

The goal, therefore, was to identify similar projects, where relevant information about the different projects was stored in a project management system. Because it was not known which similarity measures can be useful and neither which features to use, the central research question was: Which similarity measures should be used and in what way to support this knowledge management initiative. As the range of possible measures and possible configurations is large and the evaluation of each single measure and configuration is a time-consuming task, the case study lends itself to applying our framework.

5.2 Configuration of the Framework's Modules

The proprietary data within the project management system was stored in an XML dialect specific to the application that could not be imported into the framework using an existing ProMImport plug-in. This is why a new one was developed. Because it was not known which influence the granularity of the log entries would have on the similarity measure's suitability, we created the importer plug-in configurable to this respect. This allowed us to extract two, differently verbose representations. One transformed the data by interpreting the change between nine given high-level status indicators as activities. The other imported data by additionally interpreting more fine-grained interactions like "insert expense type" as activities. Having two differently large sets for the same source information supports the analysis of the effect on similarity measures that is related to the size of activity sequences. Additionally, some accounting related data was not maintained in the project management IS directly, although it is logically connected to it. Hence, in the case study setting, the imported data from the project management system had to be amended with additional data from an SAP system, for which we could reuse parts of PromImport. Implementing the new importer plugin required some effort but did not take longer than a few days. The configuration of the plug-ins however was straight forward and took only a few minutes.

One of the goals for the company was to estimate the performance of a project by utilizing the similarity of its activity sequence with respect to previously finished projects. However, the stakeholders did not know which features were the best ones to use to determine similarity, while knowing how projects as a whole can be evaluated. For this reason, making use of a supervised learning approach is a suitable approach, which justified the use of the framework's classification module. Within this step, each activity sequence was augmented with the performance judgment of the project portfolio manager using a three-valued classification indicating whether a project was positive, negative or neutral. The configuration of the classification module was straight forward and took less than an hour.

In interviews, the stakeholders agreed that the interaction between different activities on the project were related to its later performance, giving rise to the use of structure-oriented similarity measures. Given the complex interactions within a project, the company did not have an explicit interaction model for their project management system. If structure-oriented similarity measures were used, retrieving a model required using the model generating facilities offered by the framework. A limited number of algorithms included in the ProM framework proved useful in this case study. After some experimentation, the α-algorithm [2], the multi-phase algorithm [2] and the genetic-mining-algorithm [17] proved suitable enough for the model determination task.

The stakeholders could not give an informed recommendation on which properties would best support or not support a similarity determination. This is why a diverse set of different measures has been used to determine the most suitable one. As the activities in the logs amounted to changes in the project status and are known in beforehand, it was viable to interpret activities as similar whenever they have the same name. The usage of equivalence classes or the consideration of the activities' attributes was not necessary in this case. Altogether, the case study used nine similarity measures, out of which five neglected the structural properties, while the other four relied on structural properties for the determination of similarity. They included the Dice Coefficient, the Overlap Coefficient [26], a bag of words [14] adaptation to activities, a Term-Frequency-Inverse-Document-Frequency [11] adaptation to activities, the Levenshtein distance [31] for activities, graph isomorphism [27], maximum common sub-graph [5], graph edit distance [20] and random walk kernels [12]. This covered a broad range of different measures which made use of all the functionalities supported by the framework. Each of these measures has its special advantages and disadvantages which is why we expected them to operate differently well depending on the input data. However, while the description and especially the comparison of their properties is a valuable contribution, it is out of scope in this paper. The configuration of the similarity measures took no more than a couple of minutes for each measure.

The configuration of the application module was done as follows: The application that is to benefit from the determination of

activity sequence similarity is intended to estimate the performance of a project using three predefined values for the performance. To create its estimation the application does a classification, which in the case study was done using a k-nearest-neighbor classifier – a common approach for classification. To determine which neighbors, i.e. which projects are "near", the application utilized the similarity measures' results as its basis. The number of nearest neighbors typically has a large influence on the classification results and must therefore be taken into account when searching for suitable configurations. In the framework, it is configured as an incrementable parameter (see section 4.5), i.e. the framework iterates through different combinations of this parameter and tests the results of each configuration separately.

The application used a second incrementable parameter. Because the aggregation of the k-nearest neighbors' class indicator into one single answer can be done in different ways, the desired algorithm can be selected using an incrementable parameter. The application module offered four different ways to do this such as majority vote and using different weighting mechanisms according to distance. The parameter iterated over those four.

The application should later serve the purpose of providing an estimate concerning the future outcome of a current project, i.e. a project that has not ended yet. To evaluate the performance of the similarity measures and their parameter configuration, the available data was split in training and test data. The test data, however, needed special treatment. The available data consisted of finished projects, but for testing the prediction quality, it is necessary to have projects that are not finished yet. For that reason, each activity sequence was first pruned using a value as indicated by an incrementable parameter and then compared to the remaining completed activity sequences to emulate the situation of a currently running project. The pruning was performed to an increasing degree using the third incrementable parameter.

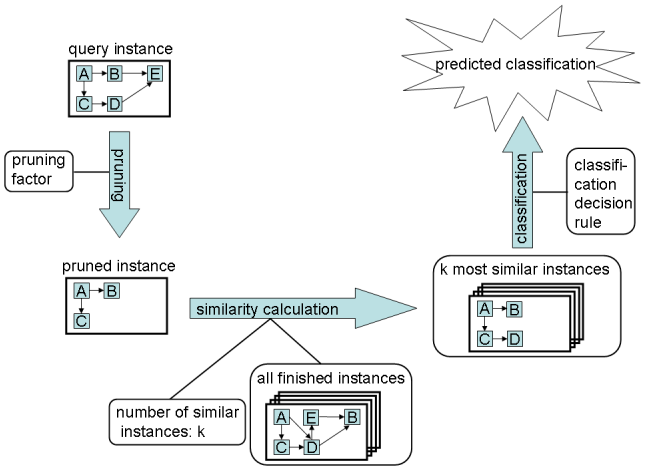

Figure 4 : Illustration of the steps for the use case

Because every project has a performance value, assigned by the project portfolio manager before the test run, it was possible to compare the results of the prediction to the actual performance value for each project. The aggregation of the single results were used to determined the overall suitability of a similarity measure for the task of predicting the performance of a project by using four different indicators that are typically used to evaluate classifiers: precision, recall, accuracy [4] and the F-measure [22]. The results were stored in a multi-dimensional data structure and were selectively displayed in a 2-D graph according to user-defined selection criteria. The steps within the case study to test the quality of different similarity measures are illustrated in Figure 4.The implementation of the application module's plug-in for the case study consumed most time and took a few days. However the program code can be integrated into the target application, therefore, the time would have been necessary anyway.

5.3 Case Study Results

The configuration as detailed in the previous section was used to perform the project performance prediction with 11 different settings for the pruning of a respective activity sequence, which reflects increasingly mature projects in terms of their run time. Also 124 unique values for the k-nearest-neighbor classifier were tested. In each iteration the four fitness indicators for the similarity measure were determined. Altogether seven different classification approaches were used, three of which were using simple heuristics[3]. The heuristics were used to compare the result of the other approaches in the light of reference results. This helped to understand the influence of potential biases in the input data. Most similarity measures outperformed all heuristics which indicated that a potential bias of the data had no significant influence. In each iteration, the data structure consisted of a 124-by-124 matrix – one line and column per activity sequence, corresponding to 15,376 entries, which in turn needed 7,688 computations of similarity values due to symmetry in the matrix. For each similarity measure, there were 11 * 124 application configurations for the 7,688 computations resulting in 10,486,432 similarity results per measure and 94,377,888 in total.

Table 1. Results of different similarity measures on prediction accuracy in the case study

	Small activity log	**Large activity log**
Measure		
Dice Coefficient	76 %	**72 %**
Overlap Coefficient	69 %	68 %
Bag of activities	70 %	70 %
TFIDF	74 %	66 %
Levenshtein	78 %	**72 %**
Graph isomorphism	67 %	33 %
Max. common-sub-graph	**79 %**	?[4]
Graph edit distance	75 %	64 %

[3] Simple heuristics were to always classify as good, bad or neutral

[4] Determination was not possible due to the algorithm's computation complexity in combination with the large dataset.

Random walk	78 %	66 %

For lack of space, not all results can be displayed in this paper. Table 1 shows the highest accuracy values for each similarity measure on both data sets. The Table shows the variation of results that can be experienced when using different measures. The difference in accuracy can be quite significant (10% on small logs, 39% on large logs), where the highest values are on a level, suitable for real life application.

The results of this case study can be seen from different angles. For the company that utilized our framework, knowing the maximum achievable accuracy for project performance prediction was valuable information, as it supports the project managers' interpretation of predictions. Without the framework, the effort for the determination would have been too high and some arbitrary, possibly non-optimal, similarity measure would have been used.

This relates to another result of the case study. We wanted to find out how well the framework could support application engineers and how much effort could be saved. The most time in the case study was spent programming the importer plug-in and the application module plug-in. These two tasks were necessary for the extension of the project management system anyway and both are independent of the similarity measures that were applied. Only the adaptation to the framework's interface caused additional effort. Together these implementation tasks took several days. Afterwards, however, the configuration of each module could be done in a matter of hours. This indicates the framework's value for application engineers, as the implementation and configuration without the framework would have taken much longer.

Another goal was to determine the scalability of the framework. In the case study, the framework extracted large volumes of data from the initial data source. Additionally, it computed process models and performed classifications. And finally, it performed nearly 100 million similarity calculations. With the exception of one similarity measure that is inherently computationally hard (it is NP-complete), the calculations were performed very quickly and none of them took longer than a few hours on standard desktop PC. This indicates the scalability of the framework.

6. CONCLUSION AND OUTLOOK

This contribution was motivated by the observation that many applications make use of the concept of similarity of activity sequences. However, the problem lies in finding the right measure for determining similarity and configuring the measure appropriately. Due to the large number of possible algorithms and configurations, a selection and configuration of a suitable measure should be automated and supported to relieve the domain specialist of routine tasks. The authors, therefore, call for the creation of a framework that supports the application engineer in finding the right measure. The requirements for such a framework are deduced by analyzing the data formats of contemporary information systems, similarity measures that are used in similarity-based applications and frameworks that are used in similar disciplines. Building upon and structuring the requirements, the components of a supportive framework are proposed. It is geared to be as flexible as possible, highlighting five modular components that allow the integration of plug-ins to cater for expendability. The framework's utility is shown in a case study where a suitable similarity measure for the performance prediction of projects is investigated. Utilizing the framework it was possible to successfully automate the computation of almost 100 million similarity values to find a suitable similarity measure. This was a task that did not take more than one person-day in the case study for configuring the framework.

While the framework was shown to be of great use, it was applied only in one case study. Great care has been taken to anticipate the needs of all applications that could potentially benefit from using the framework. To further verify the frameworks utility and also benefit from its potential, we intend to perform more case studies, especially in the area of knowledge management. We will use the framework to find suitable measures for recommendations, this time using persons and their interactions with IS as units of analysis. Another direction for further research lies in determining the properties of different similarity measures with respect to the input data. The case study already gave some interesting insights in possible properties. Those will have to be investigated more thoroughly to derive general recommendations.

7. REFERENCES

[1] Aalst, W.M.P.v.d.; van Dongen, B.F.; Herbst, J.; Maruster, L.; Schimm, G.; Weijters, A.J.M.M. (2003): Workflow Mining: A Survey of Issues and Approaches. In: Data and Knowledge Engineering, Vol. 47, 2, pp. 237-267.

[2] Aalst, W.M.P.v.d.; Weijters, T.; Maruster, L. (2004): Workflow Mining: Discovering Process Models from Event Logs. In: IEEE Transactions on Knowledge & Data Engineering, Vol. 16, 9, pp. 1128-1142.

[3] Agrawal, R.; Gunopulos, D.; Leymann, F. (1998): Mining Process Models from Workflow Logs. In: Sixth International Conference on Extending Database Technology, pp. 469-483.

[4] Baeza-Yates, R.; Ribeiro-Neto, B. Modern information retrieval.

[5] Bunke, H.; Shearer, K. (1998): A graph distance metric based on the maximal common subgraph. In: Pattern recognition letters, Vol. 19, 3-4, pp. 255-259.

[6] Dongen, B.F.v.; de Medeiros, A.K.A.; Verbeek, H.M.W.; Weijters, A.J.M.M.; van der Aalst, W.M.P. (2005): The ProM Framework: A New Era in Process Mining Tool Support. In: ICATPN, pp. 444-454.

[7] Drucker, P.F. (1999): Knowledge-Worker Productivity: THE BIGGEST CHALLENGE. In: California Management Review, Vol. 41, 2, pp. 79-94.

[8] Dumas, M.; van der Aalst, W.; Ter Hofstede, A. (2005): Process-aware information systems: bridging people and software through process technology, Wiley-Blackwell 2005.

[9] Gunther, C.; van der Aalst, W. (2006): A generic import framework for process event logs. In: Applications and Theory of Petri Nets 2005, 26th

International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings, Vol. 4103, pp. 81.
[10] Guy, I.; Jacovi, M.; Perer, A.; Ronen, I.; Uziel, E. (2010): Same places, same things, same people?: mining user similarity on social media, p. 41-50.
[11] Jones, K. (2004): A statistical interpretation of term specificity and its application in retrieval. In: Journal of documentation, Vol. 60, pp. 493-502.
[12] Kondor, R.; Lafferty, J. (2002): Diffusion kernels on graphs and other discrete input spaces, p. 315-322.
[13] Kotonya, G.; Sommerville, I. Requirements Engineering: Processes and Techniques. 1998. John Wiley & Sons.
[14] Lewis, D. (1998): Naive (Bayes) at forty: The independence assumption in information retrieval. In: Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings, Vol. 1398, pp. 4-18.
[15] McCarthy, J. (2002): Actions and other events in situation calculus, p. 615-628.
[16] McDonald, D.; Ackerman, M. (2000): Expertise recommender: a flexible recommendation system and architecture, p. 231-240.
[17] Medeiros, A.K.A.d.; Weijters, A.J.M.M.; Aalst, W.M.P.v.d. (2004): Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results. Eindhoven University of Technology, Eindhoven.
[18] Peterson, J. (1981): Petri Net Theory and the Modeling of Systems. In: PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, NJ 07632, 1981, 290.
[19] Resnick, P.; Varian, H.R. (1997): Recommender Systems. In: Communications of the ACM, Vol. 40, 3, pp. 56-58.
[20] Sanfeliu, A.; Fu, K. (1983): Distance measure between attributed relational graphs for pattern recognition. In: IEEE TRANS. SYS. MAN CYBER., Vol. 13, 3, pp. 353-362.
[21] Schonenberg, H.; Weber, B.; Van Dongen, B.; van der Aalst, W. (2008): Supporting flexible processes through recommendations based on history. In: Business Process Management, pp. 51-66.
[22] Shaw, W.; Burgin, R.; Howell, P. (1997): Performance standards and evaluations in IR test collections: Cluster-based retrieval models. In: Information Processing and management, Vol. 33, 1, pp. 1-14.
[23] Shkundina, R.; Schwarz, S. (2005): A Similarity Measure for Task Contexts. Paper presented at the 6th International Conference on Case-Based Reasoning, ICCBR, August 23-26, 2005, Workshop Proceedings, Chicago, IL, USA, p. 261-270.
[24] Spertus, E.; Sahami, M.; Buyukkokten, O. (2005): Evaluating similarity measures: a large-scale study in the orkut social network, p. 684.
[25] Stahl, A. (2004): Learning of knowledge-intensive similarity measures in a case-based reasoning. PhD, Universität Kaiserslautern 2004.
[26] Tan, P.; Steinbach, M.; Kumar, V. (2005): Introduction to data mining, Pearson Addison Wesley Boston 2005.
[27] Ullmann, J. (1976): An algorithm for subgraph isomorphism. In: Journal of the ACM (JACM), Vol. 23, 1, pp. 31-42.
[28] van der Aalst, W.; Schonenberg, M.; Song, M. (2009): Time Prediction Based on Process Mining. In: BPM Center Report BPM-09-04, BPMcenter. org.
[29] van Dongen, B.; van der Aalst, W. (2005): A meta model for process mining data, p. 309–320.
[30] Verbeek, H.; Buijs, J.; van Dongen, B.; van der Aalst, W. ProM 6: The Process Mining Toolkit. In.
[31] Wagner, R.; Fischer, M. (1974): The string-to-string correction problem. In: Journal of the ACM (JACM), Vol. 21, 1, pp. 168-173.
[32] WFMC (1998): Audit data specification. Technical Report WFMCTC - 1015 Version 1.1, 1998.
[33] Witten, I.; Frank, E. (2002): Data mining: practical machine learning tools and techniques with Java implementations. In: ACM SIGMOD Record, Vol. 31, 1, pp. 76-77.