

ACTIVE
Deliverable D2.1.1

**Construction of the probabilistic temporal process model
for knowledge processes**

| | |
|---|---|
| Editor: | Marko Grobelnik, Jozef Stefan Institute |
| Deliverable nature: | Report (R) |
| Dissemination level: (Confidentiality) | Public (PU) |
| Contractual delivery date: | 30/06/08 |
| Actual delivery date: | 30/06/08 |
| Suggested readers: | All ACTIVE project participants |
| Version: | 0.8 |
| Total number of pages: | 24 |
| Keywords: | Formalism, process model, knowledge processes |

Abstract

This deliverable defines basis for the class of models which are planned to be used in the rest of the ACTIVE project for data driven techniques. This corresponds to the bottom level of the hierarchy of models describing “knowledge processes” as defined in WP5 (Workpackage on Knowledge spaces). In brief, the formalism consists of three major components: (1) background knowledge (in the form of ontologies), (2) observed data (in the form of a stream of data items represented in different data modalities and possibly enriched with background knowledge) and, (3) objectives to optimize (providing guidelines for analytic techniques). The goal is to be able to maintain a data structure being able to store, summarize and respond to a wide variety of queries about the observed low level data and about information and knowledge derived from the process.

[End of abstract]

Disclaimer

This document contains material, which is the copyright of certain ACTIVE consortium parties, and may not be reproduced or copied without permission.

All ACTIVE consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the ACTIVE consortium as a whole, nor a certain party of the ACTIVE consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

[Full project title] ACTIVE – Enabling the Knowledge Powered Enterprise

[Short project title] ACTIVE

[Number and title of work-package] WP2: Context and Knowledge Process Mining

[Document title] Construction of the probabilistic temporal process model for knowledge processes

[Editor: Name, company] Marko Grobelnik, Jozef Stefan Institute

[Work-package leader: Name, company] Marko Grobelnik, Jozef Stefan Institute

[Estimation of PM spent on the Deliverable] 2 PM

Copyright notice

© 2008 Participants in project ACTIVE

Executive summary

This document presents an approach on modelling complex dynamic systems by modelling the data obtained from an observed environment driven by knowledge processes. Knowledge processes will further serve as building blocks of higher level applications planned within the ACTIVE project.

The main dimensions along which the approach is being developed are the type of input data the approach is able to deal with, scalability issues and background knowledge for interpretation of the observed data. The main data modalities being used are structured content (e.g., relational bases), graphs extended to networks (e.g., social networks), and textual content (document databases). The data items are coming into the system through time opening the temporal dimension when dealing with the data. For additional interpretation (e.g., data enrichment or introducing semantics) of the data we introduce databases or ontologies which further enrich the observed data. Scalability is achieved by a careful selection of appropriate analytic methods which assure appropriate time and space complexity – the approaches should cover everything from online scenarios (mining streams of data) to more batch style processing.

In the proposed approach the top most object is an environment which includes one or more observable and measurable entities whose activities we try to generalize into an abstract representation. The environment can in general be anything with some observable internal dynamics producing the data in different modalities as given above. This includes environments like personal computer desktop, e-mail client, document or web server, news forums, web 2.0 portals, social relationships of any kind, agent based systems, corporate business processes, etc. Within the environment we expect to have one, several or even a very large number of entities which interact between themselves, with a hidden or visible internal architecture and which can have interactions with the world outside the observable environment. The final goal of the presented architecture is to construct abstractions in the form of processes describing entity logs in a compressed form understandable for human reading and reusable for various machine applications. The idea is to analyse the data generated by different entities and construct a model in the form of patterns, regularities and other kinds of rules useful for different kinds of tasks like descriptive & exploratory analysis, causal analysis, anomaly detection, prediction etc. The discovered models can then be further used in higher level applications such as, for optimising personal efficiency on a desktop, for efficient communication over email or instance messenger, for competence discovery within an enterprise, for understanding of informal aspects of an enterprise etc.

Main tasks which we intend to solve with the proposed approach are the following:

- **Process description** – identifying main regularities within the processes and describe them in a formal language. Such a description can serve for better understanding of happenings within the processes and can further help observers to approach the process from different sides.
- **Causal analysis** – the goal is to be able to reconstruct causal chains of state sequences. An important subtask is root cause analysis where we want to find the trace backwards in time to find the initial cause for certain event which happened later in the sequence.
- **Anomaly detection** – identifying unusual situations (anomalies) within the processes which may be of some interest to the observer. Anomalies are usually signals for intervention and therefore an important concept to discover within the processes.
- **Prediction** – here the goal is to have a model which can predict future unseen situations. This class of models has particularly important place in the analysis of processes as on the output we get unseen situations which didn't happen yet (while usually we mainly deal with existing data).

The aim of this document is to provide detailed technical specification for the deliverable D2.1.2 (Simulator of knowledge processes through probabilistic temporal processes) which will implement most of the proposed structures with the goal to provide the first version of the software for the case studies and more general for process mining and process identification scenarios.

List of authors

| Company | Author |
|---------|-----------------|
| JSI | Marko Grobelnik |
| JSI | Dunja Mladenic |
| JSI | Jure Ferlez |

Table of Contents

| | |
|---|----|
| Executive summary | 3 |
| List of authors..... | 4 |
| Table of Contents | 5 |
| 1 Introduction..... | 6 |
| 2 Informal introduction to knowledge process mining | 8 |
| 2.1 Relation to other ACTIVE WPs..... | 8 |
| 2.2 Stream of actions..... | 8 |
| 2.3 Background knowledge..... | 8 |
| 2.4 Different levels of data complexity..... | 9 |
| 2.5 Modelling goal of the ACTIVE case studies | 9 |
| 3 Architecture Overview | 10 |
| 4 Data Preliminaries | 11 |
| 4.1 Basic Data Types | 11 |
| 4.2 Complex data types..... | 12 |
| 4.2.1 Structured records..... | 13 |
| 4.2.2 Textual data | 13 |
| 4.2.3 Networks..... | 14 |
| 5 Process modelling | 15 |
| 5.1 Process definitions | 15 |
| 5.2 Process Mining..... | 16 |
| 5.2.1 Introduction..... | 16 |
| 5.2.2 Analytic scenarios and tasks | 16 |
| 5.2.3 Analytic methods to be used..... | 17 |
| 6 Examples of process analysis | 19 |
| 6.1 Sample scenarios..... | 19 |
| 6.1.1 Desktop modelling..... | 19 |
| 6.1.2 Personal E-mail..... | 19 |
| 6.1.3 Enterprise E-Mail Server | 20 |
| 6.1.4 Contextual information delivery | 20 |
| 6.2 Case studies scenarios..... | 21 |
| 6.2.1 Cadence scenario | 21 |
| 6.2.2 Accenture scenario..... | 21 |
| 6.2.3 BT scenario..... | 23 |
| References | 24 |

1 Introduction

One of the central focus points of the ACTIVE project on the research side is modelling of informal business processes that are going on in larger or smaller enterprises. Since the usual practice in organisations is to have more or less well defined formal business processes, the informal side of an enterprise life is far from definitions which would allow formal approaches. In a way, we can say that the informal part of the enterprise's life is everything that is not regulated by formal business processes. This means that we can potentially have difficulties even describing what the informal part is not to mention difficulties when defining, monitoring and analysing what is going on within an organisation beyond the formal rules.

In the research part, the ACTIVE project deals with mainly two sides of capturing the informal part of an enterprise – from one side in a top-down manner where informalities could be described by some kind of rules and logic formalisms, and from the other side, in a bottom-up data-driven manner, where the goal is to model the processes by observing a life of an enterprise from the side where people leave their traces (in the form of some kind of log files such as, communication records, emails, documents, search logs, etc.) In this deliverable we will be concerned especially with the bottom-up/data-driven type of modelling where the goal is to perform automatic discovery of regularities in a functioning of an enterprise based on the available observed data in addition to background knowledge (designed in top-down way).

In a research and technical sense, the main dimensions we plan to take care of are: (a) scalability of the approaches for analysing enterprise data, (b) expressivity of formalism for capturing regularities in the data, and (c) ability to provide appropriate higher level symbolic summaries modelling the collected data. Special care will be given to the discovery of a temporal knowledge for the purpose of reconstructing informal processes within the measured enterprise environments.

Important guidelines for the design of the formalism are provided by (a) all three ACTIVE case studies capturing important class of process knowledge within enterprises, (b) technology workpackages requiring scalable and usable solution based on the formalism, (c) other scientific workpackages (knowledge structures, incentives, ...) taking as input results from the analytic modules, and (d) evaluation activities taking care of the big picture how the developed system performs within the enterprise.

The goal is to design a unifying formalism being able (without significant modifications) to deal with complex scenarios within enterprises and other less ACTIVE project related problems. The formalism will be designed in a way which will allow reusing of appropriate analytic methods from the areas like machine learning, data mining, and social network analysis while preserving enough expressive power of generated structures for modelling the targeted real life situations.

In particular, the aim of this document is to provide detailed technical specification for the forthcoming deliverable D2.1.2 (Simulator of knowledge processes through probabilistic temporal processes) which will implement most of the proposed structures with the goal to provide the first version of the software for the case studies and more general for process mining and identification scenarios. The results of these two deliverables (the present one D2.1.1 and the forthcoming one D2.1.2) will serve as a basis for the ACTIVE three layered architecture for modelling Knowledge Processes shown on Figure 1. The approach described here will cover mainly level 1 and provide glue between the layers.

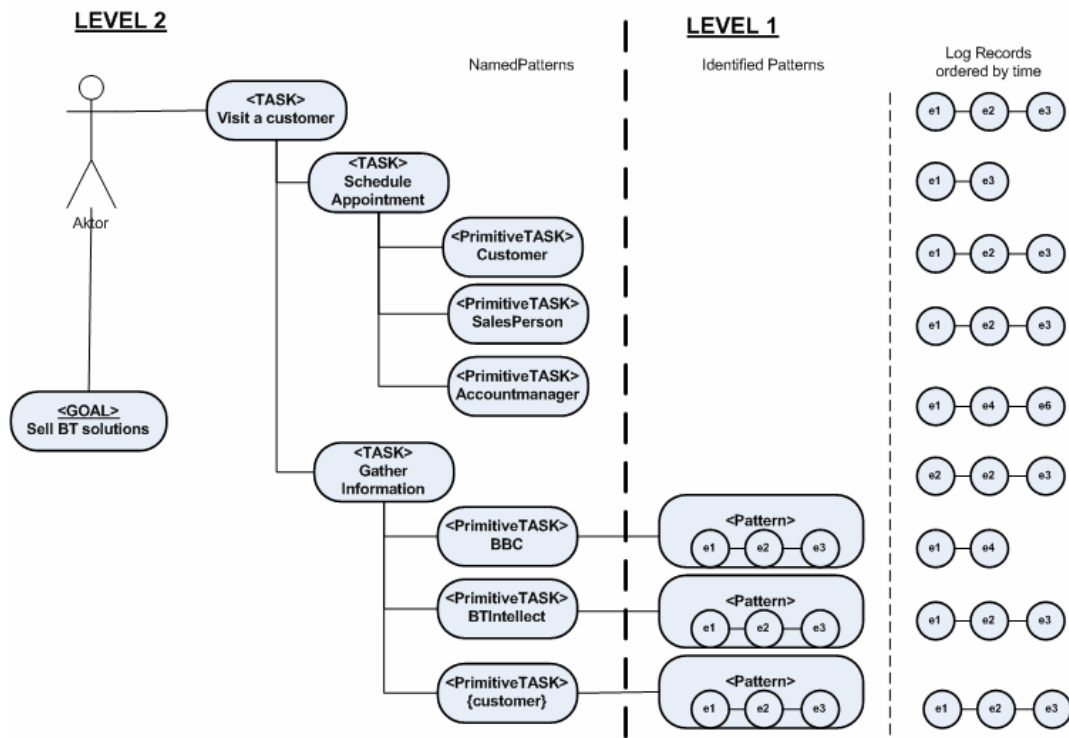


Figure 1: Three layered architecture for dealing with Knowledge Processes

In the next sections we first provide (section 2) an informal introduction into the topic of process modelling as seen in ACTIVE project, next in section 3 we give a brief overview of the architecture of the approach that is described in the rest of the document. Section 4 first provides basic data-structures followed by definitions of more complex data structures which will serve as a basis for process modelling. Process modelling is defined and described more in details in the section 5. Section 6 provides examples on how the definitions from previous sections are supposed to be used by the case studies and other examples of process modelling.

2 Informal introduction to knowledge process mining

In this section we put the deliverable in the context of ACTIVE by describing its relationship to the closely related work packages: WP5 that will be providing the data and the three case studies that are shaping and using the results. We briefly relate the low-level data and background knowledge, as expected to be observed in knowledge process mining, to the available technology. Finally we recognize different levels of data complexity (as exhibited so far by the ACTIVE case studies) that all need to be covered by the proposed approach and mention potential modelling goals for each of the three case studies.

2.1 Relation to other ACTIVE WPs

Modelling of knowledge processes as addressed here assumes that the user is a central actor in the knowledge process accessing some systems/tools and some information sources (data). Based on that, a set of possible actions is defined within WP5 and captured inside ACTIVE Knowledge Workspace developed by WP5. Event Mining Service (to be based on D2.1.2) that is a part of ACTIVE Knowledge Workspace is responsible for providing some insides to the user (by modelling the actions). It is based on probabilistic temporal process model for knowledge processes as defined in this deliverable.

Based on ACTIVE Knowledge Workspace requirements analysis (D.5.1.1) and requirements of the three ACTIVE case studies (D8.1.1, D9.1.1, D10.1.1) probabilistic temporal process model for knowledge processes will be generated from (a) a stream of actions captured by ACTIVE Knowledge Workspace and (b) relevant background knowledge.

2.2 Stream of actions

Stream of actions can be seen as a sequence of actions that are potentially connected in more complex actions forming tasks. The area of research dealing in particular with these kinds of scenarios is a subfield of Data Mining called Stream Mining (Aggarwal 2006; Gama and Gaber 2007) – the area got popular in the last 5-8 years as the amount of data increased and the usual scenario where that data was analysed in batch mode was not appropriate anymore. Analogous to Data Mining (Fayyad et al., 1996; Witten and Frank, 1999; Hand et al., 2001) Action Mining can be used to describe the observed data or for prediction. Probabilistic temporal process model, as defined in this deliverable, enables both descriptive and predictive mining. In descriptive mining, actions (described by some properties capturing nature of the observed processes) can be described in a more general or coherent way, causal analysis can be performed on them, and anomaly detection can be applied. In predictive mining based on frequent sequences of actions possibly useful patterns can be first proposed to the user to guide identification of tasks. Then each of the identified tasks can be modelled (e.g., by a task template) and applied in the future activity of the user to predict the next user's action.

In the above scenarios scalability is an important issue which needs to be addressed separately – the methods dealing with streams are typically able to take data transactions from a stream and change the resulting model incrementally with a small cost per data transaction. Generally we distinguish methods maintaining one global model of the stream data and the ones with many local models (which could act as one big model as well). In the case of ACTIVE both scenarios will be used.

2.3 Background knowledge

Depending on a specific scenario there may be some additional relevant knowledge (commonly referred to as background knowledge) available for action mining. Background knowledge can be of rather general nature (such as, an ontology that is describing possible actions) or specific (such as, description of experts skills or some relation between the experts e.g., being frequently in the same project team). The process of action mining may however require generation of background knowledge from some related data. For instance,

automatically suggesting experts for specific topics based on authorship of internal enterprise documents on the topics.

The main function of background knowledge will be to enrich the measured data from the target environment. Namely, the data we are receiving need some extra explanation which is not part of the measurements – additional information/knowledge in the form of databases, rules, ontologies etc. should therefore serve mainly as data interpretation facility and a way how to bring semantics into the measurements. For example, in practice this can mean extending the existing data records coming from measurements with extra features (based on background knowledge) which will be used for building better statistical models.

2.4 Different levels of data complexity

Probabilistic temporal process model for knowledge processes is defined here to cover different levels of data complexity where time is an inherent property of the data regardless its complexity. By data complexity we mean here data ranging from simple record of action over records equipped with additional textual description to network of records (where both records and links can have additional textual description). The resulting class of models (modelling temporal processes) is based on an underlying data structure that enables capturing information provided as text, network and time (TNT).

In the basic setting the data structure is a timeline of simple records where the main goal is to find patterns leading to identification of tasks that the user is performing and using the identified tasks to predict/suggest to the user the next action (system and/or information source that the user might like to access next). For instance, in WP10 Cadence VCAD Services case study requires automatic discovery of patterns within processes.

However the timeline of records can be a timeline of documents where the underlying data structure should enable compact storage and representation of text to support search and visualization. In WP8 Accenture case study requires support to enterprise search over a pre-tagged collection of documents.

Moreover, the records timeline can contain links between the records based on some record properties forming a dynamic graph/network. In that case mining can be performed on nodes of the graph and/or links of the graph (e.g., taking into consideration neighbourhood of a node). For instance, in WP9 BT Global Services case study the users are experts that are working in smaller teams simultaneously on several topics with a need for agile work and knowledge transfer.

2.5 Modelling goal of the ACTIVE case studies

We have already mentioned that action mining can be used for description or prediction and we expect that some scenarios of the ACTIVE case studies will call for one or the other.

Description can involve visualization of actions at different levels of complexity (basic actions, sequences of actions - patterns, tasks), grouping of similar actions, or identification of frequently occurring sequences of actions. These can be useful for automatic comparison of patterns in the processes with respect to a predefined workflow required in WP10 Cadence VCAD Services case study and in the other two case studies when requiring discovery of patterns and composing patterns into tasks.

Prediction on the other hand can be seen as prediction of the next action where we define different level of complexity based on the action requiring system/tool and/or information source. In WP9 BT Global Services case study the users are accessing several systems/tools and several information sources meaning that prediction of the next possible action should consider suggesting to the user both a system and an information source (data). In WP8 Accenture case study the focus is on using enterprise search where the user is working with single a tool but accessing numerous information sources. In WP10 Cadence VCAD Services case study we expect that the emphases is not on using several systems or different information sources, so we can say that the action is connected to the usage of a single system and a single information source thus there is no need for predicting the next system or data to be requested by the user.

3 Architecture Overview

The proposed approach to the process modelling is shown on Figure 2. The top most object is an environment (“**System**”) which includes one or more observable and measurable entities (“**Entity**”) whose activities we try to generalize into an abstraction (“**Process Model**”).

The environment “**System**” can be in general anything with some observable internal dynamics. This includes environments like personal computer desktop, e-mail client, document or web server, news forums, web 2.0 portals, social relationships of any kind, agent based systems, corporate business processes, etc. It is not necessary to have a complete understanding and observation of the environment – we should acknowledge that there can always be some hidden parts having influence on the dynamics within the environment.

Within the environment we expect to have one or more (potentially a very large number of) entities (of type “**Entity**”) which interact between themselves, with a hidden or visible internal architecture and can have interaction with the world outside the observable environment. Each entity should have a well defined state which captures relevant measurable information about the activity – state of an entity in general cannot capture everything needed for complete understanding of the dynamics, but rather what is practical to measure and analyse. A state in general consists of a set of variables representing different aspects of the measured entity. Each entity has a log of past activities represented as a sequence of past states which serves for further analytic purposes. Examples of entity logs are mouse movements, activated applications, incoming and outgoing emails, web server log files, news forum postings, instance messenger logs, etc.

All the entity logs include lots of information which is unmanageable without proper abstraction. The goal of the presented architecture is to construct abstractions in the form of processes describing entity logs in a compressed form understandable for human reading and reusable for various machine applications. The idea is to analyse the data generated by entities and construct a model in the form of patterns, regularities and other kinds of rules for different kinds of tasks like descriptive & exploratory analysis, causal analysis, anomaly detection, prediction etc. The discovered models can then be further used in higher level applications for e.g. optimising personal efficiency on a desktop, for efficient communication over email or instance messenger, for competence discovery within an enterprise, for understanding of informal aspects of an enterprise etc.

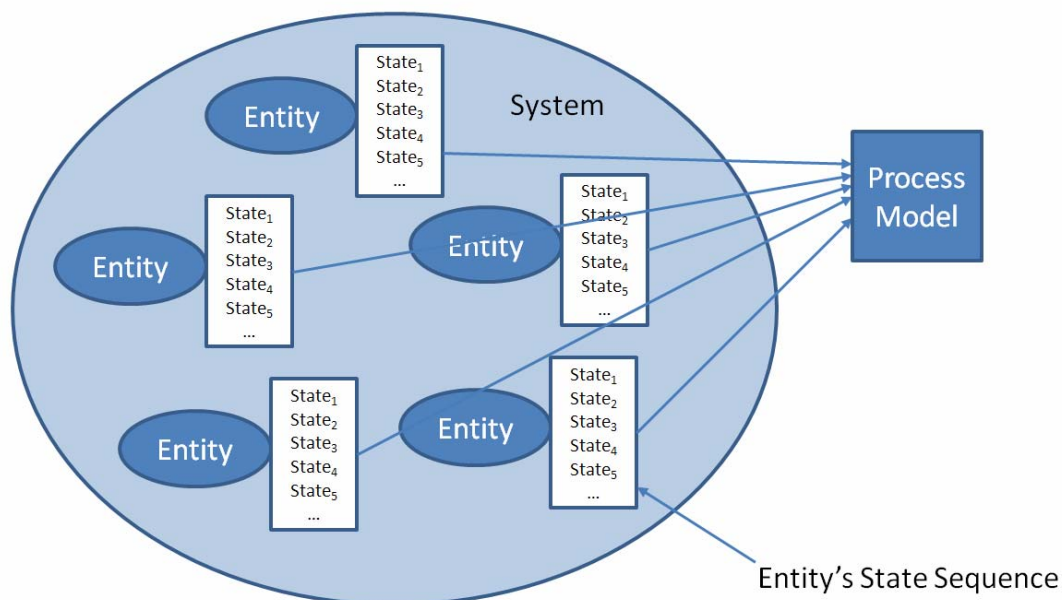


Figure 2: Architecture of the process modelling.

4 Data Preliminaries

The key element for modelling knowledge processes is the availability of data within enterprises (where project case studies are just a good sample of situations on the market). In the ideal case, the data would capture all the activities on different levels of an organisation which would allow reconstruction of most processes and other relevant concepts to understand how the organization is functioning. But unfortunately, this is impossible to achieve for many practical reasons. After some review of what kind of data is realistic to retrieve (like availability) and what type of data is manageable (like data complexity, scale, granularity), from technical point of view, we concluded to use some of the fundamental data types on which we will define characteristic operations.

The data we can expect within the project and other similar situations can be of different types (modalities). The most typical data types are the following:

- Structured records (e.g. databases)
- Textual data (text documents with some or without structure)
- Networks (set of connected objects and their relationships)

Each of the above data modalities allows characteristic set of operations which could be used for extracting higher level constructs from data. In the following sections we will go briefly through each of them.

Each of the data types has inherent temporal nature, which means that structured data records and documents are coming in streams and networks have dynamic nature. These features will be specified more in details in the following sections.

Next, we will define the data model for the structures we expect to be dealing with within the project. The data model will be defined in a way which will allow relatively straightforward implementation in procedural and declarative programming languages. The data model will be described as a set of production rules with additional textual explanations.

4.1 Basic Data Types

Any data can be of atomic or structures types:

DataType = AtomicDataType | StructuredDataType.

In the text we will refer to **DataType** with symbols **T**, **T1**, **T2**, ... **T_i**.

Atomic data types are the ones known from programming languages – in the description below we won't go into the details of lexical representations for each of the atomic data types but rather stay on the level of informal description:

AtomicDataType = Void | Boolean | Integer | Float | String.

Each of the atomic data types has defined set of operations which are well known from algebra and programming language practice.

Structured data types are composed from the atomic data types and other structured data types. The basic ones which we will define here will be used in the definition of the more complex structures in the following sections.

StructuredDataType = Pair | KeyVal | Tuple | Vector | Stack | Queue | Set | Map.

All of the structured data types require additional subtypes which need to be specified when the data type is getting instantiated.

The simplest of the structured data types is simple pair of two data types.

Pair<T1, T2 > = pair of two values <V1, V2> where V1 is of type T1 and V2 of type T2.

The only operations defined on the pair are accessing of the sub-values and comparison of the types.

KeyVal type is an upgrade of the **Pair** consisting from the identifier (name) of type **String** and a value of an arbitrary type. **KeyVal** is used for construction of more sophisticated data types where we usually assume some indexing schema over identifiers to access the values.

KeyVal<T> = **Pair**<**String**, T>.

Fundamental data type which corresponds to the usual notion of a data-record when dealing with databases is **Tuple** – it consists from a fixed set of **KeyVal** pairs which corresponds to fields in a data record. Tuple has certain number of components, where each of the components has different identifier. Once being defined, a value of the type **Tuple**, cannot be changed in the number of components.

Tuple<T₁, T₂, ... , T_i, ... , T_{max}> = fixed set of <V₁, V₂, ..., V_i, ..., V_{max}> where V_i is of type **KeyVal**<T_i> and each of the sub-values in V_i is retrievable by a string quantifier of **KeyVal**.

The first structured type of the variable length is **Vector**, which includes series of values of a same type T. Number of operations are defined on the **Vector** type, including adding and deleting values within a vector.

Vector<T> = series of values <V₁, V₂, ..., V_i> where all the values V_i are of the type T

Important variants of the data type **Vector** are data types **Stack** and **Queue** which implement in an efficient way slight different types of operations (FIFO, LIFO principles) which are well know from the algorithmic literature.

Stack<T> = **Vector**<T>

Queue<T> = **Vector**<T>

Last two of the fundamental data types are **Set** and **Map** which correspond and get implemented by the class of data structures like hash table, search trees and other indexing structures. The key property of these structures is efficient access and update operations for data records (typically in constant or logarithmic time). The main difference between Map and Set is the fact that each value (of type T1) in **Map** carries also a corresponding value (of type T2) while in **Set** we operate just with a set of values (of type T).

Map<T1, T2> = {<V_{1,1}, V_{1,2}>, ..., <V_{i,1}, V_{i,2}>}, a set of pairs where T1 denotes the key and T2 the corresponding value.

Set<T> = {V₁, V₂, ..., V_i}, a set of values of type T.

4.2 Complex data types

Complex data types consist of basic data types and implement the key data structures for monitoring processes within enterprises. We tried to restrict the selection to the minimal possible set of data types covering typical data modalities one can find in enterprise environments. We identified three major types of data which we use in a static and stream context (meaning either we have data available as a static dataset or the data is coming in a stream).

- Structured records as found in databases represent the most basic way of representing data in an environment. Typically all the standard data of an enterprise would be represented in this way – this

corresponds to the data stored in relational databases, indexed in various ways and managed by DBMS type of systems.

- Textual data with potentially some additional structure (in a form of e.g. documents, web pages, presentations) represent most of the unstructured data retrievable from enterprise document servers. This type of data represents core of the less formalised part of the enterprise data. By extracting data from text and putting it into other structured forms (like structured records or networks) we make steps towards manageability of the less formal information.
- Networks are the most generic formalism to describe structure in the data. The basic underlying structure for a network is a graph with additional information attached to the vertices and edges of a graph. By structuring the information in different ways on a network we can represent very complex situations in an environment. In addition to the standard definition of a static network we will define also a dynamic version of it which changes through time which will represent basis for monitoring processes within an enterprise.

4.2.1 Structured records

Structured records correspond to the basic data type **Tuple** which consists from a fixed set of **KeyVal** pairs.

Record = Tuple

Operations defined on records are mainly restricted to comparisons and accessing its components.

In the scenario where we are getting a stream of records, we operate with the structure where retrieved vectors are stored in a **Vector** or a **Queue**, enabling various kinds of analysis.

Vector<Record> = storage of all the records coming from the stream

Queue<Record> = storage of only the last window of records coming from the stream

4.2.2 Textual data

Textual document without any additional structure is most typically represented as a “bag of words” which effectively corresponds to a vector of words from the documents and their frequencies (or some other type of weights like TFIDF or similar). Such a representation on a database of documents allows performing a series of scalable linear algebra analytic operations on a document set. This representation is a basis for today’s research fields like Information Retrieval and Text Mining.

Document = Vector<Pair<String, Float>> document is represented by a set of words with corresponding weights.

In the stream scenario where the documents are coming in time, we use similar representations as for structured records where we deal either with complete set of documents or with the most recent ones.

Vector<Document> = storage of all the documents coming from the stream

Queue<Document> = storage of only the last window of documents coming from the stream

Another possible representation for a text document is a set of triples of the form <Subject, Predicate, Object>, where each of the constituents of the triple is a word or phrase from the document. In more advanced form, the pronouns get replaced by proper names to which they refer. This type of document representation is less commonly used in the research and even less within the commercial settings although the recent developments (like OpenLinkData initiative, PowerSet.com Company) show the opportunities of such a representation. Within the ACTIVE project we plan to use this type of representation in the cases where textual documents will get involved (like Accenture or BT case studies). Within this formalization we

will avoid detailed specification – the closest data structure supporting this representation is a network presented in the next section.

4.2.3 Networks

Networks are by their basic nature graphs with some additional information attached to the nodes (vertices) and/or links (edges). The basic graph structure (without any additional information) is represented as a set of vertices and edges which connect nodes (for simplification, we will define here only directed graphs)

Vertex = Integer, where the number represents the identifier of a vertex.

Edge = Pair<Vertex, Vertex>, where components represent source and destination vertices.

Graph = Pair<Set<Vertex>, Set<Edge>>, graph is a set of vertices and edges.

As mentioned, the main difference between networks and graphs, as noted in the literature, is additional information which is attached to the graph elements (vertices and edges). Therefore, we use instead of the term “graph” the term “network and instead of the terms “vertex” and “edge” the terms “node” and “link”.

In the next definitions we will define **Node**, **Link**, and **Network** where the main difference from the definition of a **Graph** is additional information attached to its elements.

Node = Pair<Vertex, Tuple>, where Node is Vertex with attached data in Tuple

Link = Pair<Edge, Tuple>, where Link is Edge with attached data in Tuple

Network = Pair<Map<Vertex, Tuple>, Map<Edge, Tuple>>, network is a map of vertices and edges with attached data

Dynamic network is a variant of a network where, the structure of the network changes through time (nodes and links are getting added and removed) and the data attached to the network is changing. Specifics of a network dynamics is defined by the application. One way how to formalize the dynamics of networks is representing dynamic network as a sequence of networks $\langle \mathbf{Network}_t \rangle$ where the **t-th** element is a network at the time **t**. Out of such structure (which is usually too expensive to represent directly) it is possible to extract temporal analytic information. Structure with the same characteristics, which could be easier to implement is the following (**Tuples** being extended with the temporal window):

DynamicNode = Pair<Vertex, Queue<Tuple>>, where Node is Vertex with attached temporal window of data

DynamicLink = Pair<Edge, Queue<Tuple>>, where Link is Edge with attached temporal window of data

DynamicNetwork = Pair<Map<Vertex, Queue<Tuple>>, Map<Edge, Queue<Tuple>>>, network is a map of vertices and edges with attached temporal window of data

In a structure like **DynamicNetwork** one can represent very rich set of scenarios which include also the ones from ACTIVE project. By a proper parameterisation of the structure through extra information on nodes and links it is possible to represent scenarios which include textual content (repositories of documents), social networks (e.g. relationships between the people), dynamic of an environment (how things develop through time) etc. The structure is interesting also from analytic point of view since allows several types of analytic approaches – more of those we will discuss in the next sections.

5 Process modelling

5.1 Process definitions

A “process” is often used as an abstraction for a sequence of events. Processes with some particular meaning would often get a special name denoting the effect the characteristic sequence of events has on the environment. Different contexts where the notion of process has its meaning are well described on <http://en.wikipedia.org/wiki/Process>.

A process always happens to an entity **Entity** which operates within a system **System**. **Entity** is described by its state **State** which is usually described by a set of variables of a type **Tuple** denoting various characteristics of the observer **Entity**.

State = Tuple, where variables within a Tuple describe state of an Entity.

A process is a sequence of states in which the observed entity appears. In the case of discrete time (being reasonable assumption in our case) we can say that the sequence of states is a **Vector** of the type **State**

StateSequence<State> = Vector<State>, where a the states are order in time

Each state in the sequence is derived by the following formula:

StateSequence[t=0] = initial state where state variables get initial assignment of values.

StateSequence[t+1] = NextState(Process[t], Action, BKnowledge).

Where an **Action** can be either an empty value or of a type **Tuple** generated by an external source within the **System**. Typically, an **Action** could be an asynchronous event which influences the state of the observed **Entity**.

Action = Tuple.

Background knowledge represented by the parameter **BKnowledge** serves as an additional (mainly static) resource for interpretation of the **Process**, **State** and **Action** within the function **NextState**. **BKnowledge** can appear in the form of a database, set of rules or even as an ontology providing semantics for the data being involved in the description of states within sequences and further on within processes. In this formalization we will avoid specifics of the background knowledge – each example of the **NextState** function in the next sections will provide its own specific type of additional information needed to describe the states and processes. Operationally **BKnowledge** can be understood as a generic query function returning values on the requested (domain specific) queries.

Having the data collected in the form of one or several **StateSequence** structures (e.g. one for each **Entity** within the **System**) we can build a model of the data by using analytic techniques such as statistics, machine learning, data mining etc. The model is expressed in a formalism (hypothesis language) supported by the selected analytic method (like rules, decision trees, linear or non-linear functions, etc.). The purpose of the model is to abstract or summarize the analysed data into a shorter description which typically probabilistically resembles the structure and properties of the data. Specific type of modelling and selection of the analytic methods depends on the task (described in the next section). The data object we are constructing from **StateSequence** data is **ProcessModel** and is constructed through modelling process using **BKnowledge** and **Target** value:

ProcessModel_{Target} ← Modelling(AnalyticAlgorithm, Vector<StateSequence>, BKnowledge, Target)

ProcessModel is further used in any situations as independent functions getting on the input data from a **StateSequence** and returning the **Target** value which were modelled within the **Modelling** phase:

Target ← ProcessModel_{Target}(StateSequence, BKnowledge)

5.2 Process Mining

5.2.1 Introduction

Analysis on the top of the event sequences can be done in various ways. Many areas of science are actively working in subfields of analysis of processes which are mainly determined by the structure and properties of the **State** description of the observed **Entity** and by the language used to describe **NextState** function.

First example of process modelling is from mathematical analysis. Differential equations are well know mechanism for describing processes for the cases where the **State** is described with continuous variables and the function **NextState** is described in analytical language. In the same way, but on the other side of the spectrum within the areas of data mining there is a family of algorithms on “Mining Frequent Episodes” (Mannila, Toivonen, Verkamo 1995) which operate on a set of binary variables and the language for describing **NextState** function is “association rules formalism” (if-then rule).

An important class of processes, which will be of particular interest for our work, are situations where the **State** encodes a network (of a type **Network** or **DynamicNetwork**) and by the change of state (function **NextState**), the observed **Entity** moves from one node to another. Apart from encoding a **Network**, a **State** can include in the rest of its variables also other characteristics of the modelled **Entity**. This is entirely domain dependent and can cover wide range of situations. The importance of this class of processes is relation to process diagrams which are widely used within business process modelling. Diagrams, which can be easy represented by an instance of a type **Network** and corresponding **NextState** function can describe broad class of situations which we can envisage within the ACTIVE project.

5.2.2 Analytic scenarios and tasks

Two typical scenarios for analysing processes are:

- Analysing the data appearing within the **StateSequence** sequence. This allows finding regularities within the sequences without any specific assumptions on the structure of data etc. This kind of scenario allows mainly understanding of the process event sequences but not other tasks.
- Identification of the **NextState** function is the most common approach to the analysis of the process data. The goal is to capture specifics of the process within the model by which we describe the function. For certain classes of models (defined typically by the language used for the description of the **NextState** function) we can efficiently calculate the function while for some other classes it can be much more difficult or almost impossible. Usually we are trading between the computational efficiency of the reconstruction of the function versus expressivity of the language for describing **NextState**.

Main tasks which we try to solve by process modelling are the following:

- **Process description** – here the main goal is to identify main regularities within the processes and describe them in a formal language. Such a description can serve for better understanding on what is going on within the processes and can further help observers to approach the process from different sides.
- **Causal analysis** – the goal in this task is to be able to reconstruct causal chains of state sequences. An important subtask is root cause analysis where we want to find the trace backwards in time to find the initial cause for certain event which happened later in the sequence.
- **Anomaly detection** – in this task we want to identify unusual situations (anomalies) within the processes which may be of some interest for the observer. Anomalies are usually signals for intervention and therefore important concept to discover within the processes.
- **Prediction** – here the goal is to have a model which can predict future unseen situations. This class of models has particularly important place in the analysis of processes because on the output we get unseen situations which didn't happen yet (while the other situations deal mainly with the existing data).

5.2.3 Analytic methods to be used

On the top of the structures defined in the previous sections like **Process**, **StateSequence**, **DynamicNetwork**, **Network**, **Document**, etc. one can use a broad class of methods from research fields like machine learning, data/text/web mining, statistics, time series analysis, social network analysis, computational linguistics, etc. In particular, the methods dealing with temporal data would be of particular interest for some of the problems related to ACTIVE project tasks.

Following the structure from the previous section, we can assign some more relevant analytic methods to solve each of the tasks. The assumption here is to have on the input the structure **StateSequence** (possibly enriched with **BKnowledge**) upon which we execute some of the analytic methods to solve the task. Most of the tasks deal with the reconstruction of the function **NextState** or with analysis of **StateSequence** data.

Process description – here the goal is to find the structure within the data for the purpose to understand, summarize, and characterize the structure within the **StateSequence** data. In other words, we would like to explain and understand the functioning of the function **NextState** operating in the typical situations described in the **StateSequence** data structure. Main classes of methods for decomposing the **StateSequence** data into some kind of structure are so called unsupervised methods like clustering and other eigenvector decompositions. These kinds of methods provide insights into the structure of the data and together with additional domain knowledge being used for exploratory data analysis enable deeper understanding on what is going on within the data. These methods are often an input for data visualization methods which typically generate graphical summaries of the data.

An example of such descriptive analysis would be analysis of a corporate email server an approximate organigram of the analysed organisation based on the sub-communities discovered in the analysis (Figure 3). Data in this application represent records from email server on emails to whom within an organization – the data are taken from “Jozef Stefan Institute” email server – the method used was hierarchical divisive K-Means clustering (Manning, Schütze 1999). The work had been done within the FP6 SEKT project.

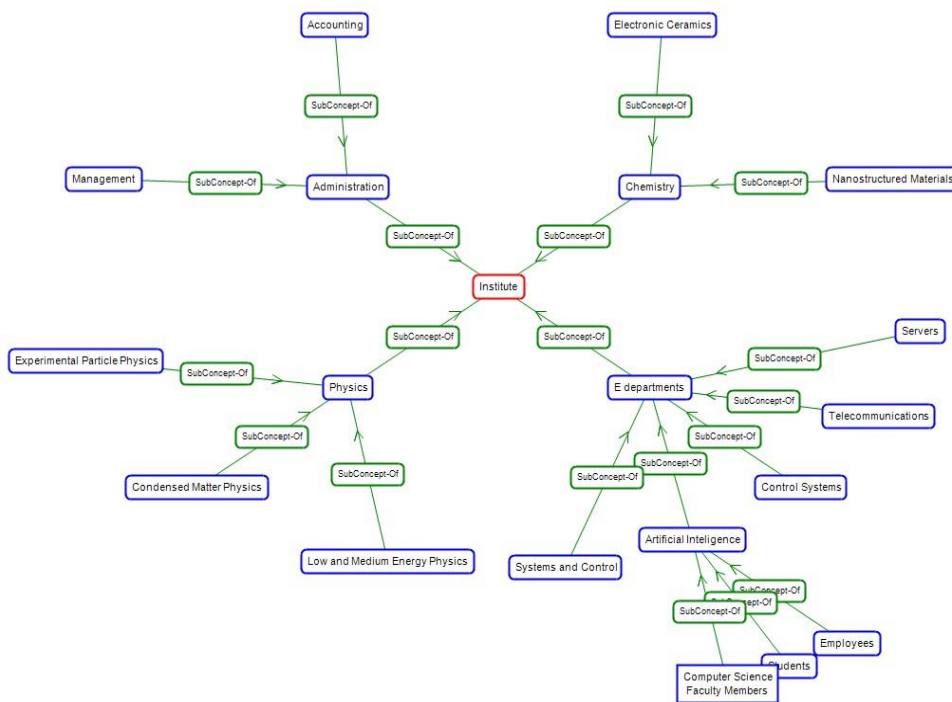


Figure 3: An approximate organigram constructed semi-automatically based on the 2.7 million email transactions from Jozef Stefan Institute.

Causal analysis – for this task we typically need to construct inverse of the function **NextState**, where the goal is to predict which state was preceding the current state. First, we need to construct (either from data or domain knowledge) set of discrete states in which the process can appear. Further, with the analysis of the past data of process behaviour we construct models of relationships between the discrete states which enable causal analysis. Typical scenario we try to solve is given an event, causing the process to appear in a certain state, to generate list of other states which most likely preceded our current event to happen. The key here are the methods to detect relationships between the states – this can be done in various ways (depending on the problem we are solving). The most obvious one (when no extra domain knowledge is available) is correlation between states and events that happen where correlation represent probabilistic relationship between the states. In the presence of additional information we can build local predictive models (with e.g., Support Vector Machines or similar) for “predicting” which path to follow from a state backwards to the cause – for such a predictive tasks we need some labelled data for initial training. Another option is also to build global model of relationships via Markov models (like Hidden Markov Model) where we can test how likely a certain sequence of actions could have happened (Manning, Schütze 1999).

Example of causal relationships are probabilistic Markov-network style if-then rules which establish probabilistic relationship of preceding events based on the later sequence of events.

If Device-123 failed, then possible causes are:

preceding failure of Device-456 (with probability 0.7)

preceding failure of Device-567 (with probability 0.2)

preceding failure of Device-789 (with probability 0.1)

Anomaly detection – for this task we generally apply statistical measures for measuring surprise or probability of a certain sequence or an event to happen. In a general case, we compare a particular probabilistic distribution with a general one (e.g., using chi-square) and if it is statistically significantly different then we report an anomaly. With an additional domain knowledge which provides clearer model of behaviour of the observed system we can detect anomalies in a more accurate way. Anomaly detection is in general a very hard problem because we try to detect something which is unusual (irregular), but with analytic methods we can detect only certain class of irregularities which occasionally might not be sufficient for what a human end-user would consider and classify as an anomaly.

Example of anomalies in the scenario of monitoring computer’s desktop activities of the user could be unusual usage of the resources based on the running applications. Unusually high usage of certain resources could be an anomaly caused by a virus or malfunction of the application. Anomalies are often reported as comparison between what is usual and in what way the anomalous situation differs from that.

In the application of observing corporate business processes (being typically in the form of process diagrams) an unusual increase of activities in the part of the diagrams where intensity is not as high. This kind of information could be a signal for management to react in a proper way.

In the application of observing corporate e-mail server, it could be a change in the intensity of communication between some project group members which can have a cause in problems in their relationships which can further lead to problems on the observed project.

Prediction – here the goal is to predict which state will be the next one in the sequence of states within the **StateSequence** structure. In this task we typically take a window of recent states and try to predict most probable next state. The models we build for prediction are typically built locally for each state in the vocabulary of states. To solve the task we have available a broad range of predictive modelling methods from machine learning and data mining. The methods being used the most in the recent years for predictive tasks are Support Vector Machines (SVM) (Cristianini, Shawe-Taylor 1999) and Conditional Markov Fields (CRF). (Lafferty, McCallum, Pereira, 2001)

An example of prediction can be within the application on personal e-mails where the goal is to predict when we might expect a response to our e-mail sent to a certain person.

6 Examples of process analysis

In the following sections we describe mappings of the ACTIVE project case studies to the process model analysis as described in the previous sections. Since not all the details of the case studies are known yet, these descriptions should be understood as preliminary – the final scenarios will get know once the case studies data will be available – at that stage very precise mappings to the process model will be possible.

6.1 Sample scenarios

In the following subsections we will show how the most important variables, from the model we defined in the previous sections, could be instantiated to real-life categories. The aim is to show how the proposed approach could be used for a broad spectrum of applications related to process modelling.

6.1.1 Desktop modelling

Modelling activities on the end-user's desktop is one of the primary tasks aiming to analyse how the user is interacting with the operating system, applications and internet – the goal is to use this information for better information delivery and to optimize the user's interaction with the desktop. Projects with similar goals are FP6-IP NEPOMUK and Microsoft's project named Research Desktop. The following items show mappings of desktop modelling task to the concepts within the proposed framework.

- **System** = computer's desktop
- **Entity** = computer's application possibly instrumented to extract additional internal information. Within the ACTIVE project the plan is to instrument a selected set of applications with specially prepared software plug-ins – these will generate detailed information about internal states of the observed applications.
- **State** = description of a particular application where individual fields within the state describe internal states of the observed application (especially when the application is instrumented). For instance, if the observed application is internet browser, the state could include URL and text of the opened page.
- **NextState** = function generating new state record for the observed application – typically would be triggered by an **Action** initiated by the user or by the computer itself. In the example of internet browser the **Action** could be opening a new Web page, in the example of e-mail client the Action could be receiving, sending or reading an email message.
- **StateSequence** = log of activities for a particular application
- **ProcessModel** = **DynamicNetwork** consisting of states describing internal states of the all observed applications – this will allow modelling dynamic dependencies within individual applications and between different applications.

6.1.2 Personal E-mail

The task of analysing personal email is one of the fundamental tasks for organising end-user's communication with its environment and can significantly improve efficiency of performing tasks communicated over the email. Related commercial project is from the company Xobni.com.

- **System** = Personal E-mail client having access to the collection of all user's emails.
- **Entity** = E-mail address of one of the entities (typically people, but could be also mailing lists, bots etc) the user is communicating with

- **State** = state of communication with the **Entity** – this can include simple statistics (like in Xobni), social network, content, state of discussions etc.
- **NextState** = change of a state triggered by one of the characteristic e-mail events like receiving, submitting, deleting, tagging, storing of an e-mail
- **StateSequence** = log of interactions between the user and the **Entity**
- **ProcessModel = DynamicNetwork** consisting of interlinked **Entities** for which the system is able to extract social dynamics of interaction, topic structure and dynamics – the goal is to use this kind of information for helping the user when responding to e-mails or to have an overview over personal communication activity.

6.1.3 Enterprise E-Mail Server

Analysing enterprise e-mail server enables top level view over the dynamics within an enterprise. It enables reconstructing social and topic structure through time which further enables to reconstruct processes within the enterprise (at least the ones observable through email communication). It is important to say, the main problem with this type of task is the availability of such data because of the sensitivity of the information being involved. Most of the work has been done on the Enron e-mail corpus being the only corpus of the kind available for the research.

- **System** = Corporate E-mail server having access to the collection of all corporate emails.
- **Entity** = E-mail account registered within the enterprise e-mail server
- **State** = state of communication between the **Entity** and the rest of the network (internal and external to the enterprise) – this can include social network, content and temporal information
- **NextState** = change of a state triggered by one of the two main e-mail events: submitting and receiving.
- **StateSequence** = log of interactions between the **Entity** and its e-mail social network
- **ProcessModel = DynamicNetwork** consisting of interlinked **Entities** for which the system is able to extract social dynamics of interaction, content contexts of interaction – the goal is to use this information to build abstract models of the observed enterprise in the form of reports on social and content trends, when using background knowledge about the enterprise organizational structure one can model interaction at different levels of enterprise units and extract process structure in the form of probabilistic process diagrams

6.1.4 Contextual information delivery

Contextual information delivery can have many specific applications – in this case we will concentrate on internet search and browse scenario. The goal is to model the user's interests and task contexts by observing the users' activities – this would lead to a construction of a personalized and contextualized document search ranking function which can be used on the top of internet search, intelligent browsing suggestions, contextualized browsing history etc. Similar goals had FP6 SEKT project (a prototype system SEKTbar) and FP6 NEON project (a prototype system Search-point)

- **System** = Internet browser
- **Entity** = Topic/context automatically or semi-automatically identified by the **NextState** function.
- **State** = state of a topic – typically this would include set of pages and documents viewed by the user
- **NextState** = change of a state triggered by new document being view by the user – side effect of the **NextState** function is also construction of new **Entities**

- **StateSequence** = log of documents viewed by the user on the topic/context represented by the **Entity**
- **ProcessModel** = **DynamicNetwork** consisting of interlinked topics – based on such a network one can construct customized document ranking functions

6.2 Case studies scenarios

6.2.1 Cadence scenario

In the Cadence scenario there is a clearly defined goal which must be achieved through the sequence of transformations from the beginning empty state to the final state. The goal in the Cadence scenario is always building of a product where we start with an informal specification and end with a finalized product. To achieve the goal we have a set of well defined transformations mainly in the form of software modules taking on the input the product in more preliminary stage and giving on the output the product in the more advanced stage. The set of transformations define the graph of states and transitions between them. Some transitions between the states are deterministic and some nondeterministic. The task for analytic methods would be to observe the past logs of the Cadence processes and to build a model for states to reduce the non-determinism when transitioning the state graph.

Formally, we can define a state of a product as a **Tuple** of a type **State** where the **Tuple** includes description of the product. One of the variables within the **State** encodes also the current position within the state graph. State graph (graph of states and possible transitions) is otherwise stored in the form of the background knowledge within the **NextState** function. State graph consist from:

- Start-state where the state of the product get initialized to the starting set of values,
- End-state denoting the set of variables describing the product reached their final stage,
- Intermediate-states denoting intermediate stages of a product – each of the intermediate states has one or several possible transitions to the next states denoting application of one of the predefined activities for developing the product.

The key for efficient development of the product is well designed **NextState** function which takes on the input state of the product, decides for one of the transitions and returns the state of the product in the more advanced stage. This can include deterministic or nondeterministic decision making including human decisions.

ProductState_{t+1} = NextState(ProductState_t)

The main tasks within the ACTIVE project is to construct the function **NextState** from the past product development logs – special focus will be given to the models for deciding which path to following from the states with multiple follow-up transitions. In general all four types of tasks (description, causality, anomalies, prediction) as defined in the previous sections could be of interest.

6.2.2 Accenture scenario

In the Accenture case study the main goal is to understand the functioning of the enterprise with the purpose to increase knowledge sharing between consultants. On the input we have large repository (approx. 300 000 units) of documents written through time. The documents include topics on which the consultants were working at particular time and include also names of the people involved and several other meta-data. Apart from the documents we have available also background knowledge in the form of the listing of Accenture consultants, organizational structure, consultants' search logs and potentially some other data which can help for modelling the enterprise.

As specified in the ACTIVE deliverable D8.1.1 within the Accenture case study there are three main sub-problems where in our understanding the main one (which is the prerequisite for the other two) is the one addressed as “*Non-task specific Knowledge Inquiry*” with the goal to capture and model the complexity of an enterprise like Accenture with no specific task in mind. To approach the generic model of an enterprise based on the data we have available for the case study our strategy is to perform the steps described in the next paragraphs.

Given the corpus of documents of the type **Vector<Document>** we can perform the following pre-processing steps which will bring us to the stage which will allow solving several other tasks.

1. From the documents in the repository we can extract named entities, in particular the names of people which appear within the documents and which are Accenture employees. The names of employees and other named entities will be in the next stages organised in the **DynamicNetwork** with relationships as happen in time (since also the documents appear through the time).
2. Since the documents in the database include a textual content we can identify the topics which Accenture is working on – furthermore, since the documents appear in time, we can identify the evolution of the topics through time and consequently detect topic trends.
3. Within the first step of this pre-processing of documents, we extract the names of the people appearing within the documents. At this stage, based on the text where the names are appearing, we can assign topics to each of the names appearing within the documents. By doing this, we create topic profiles for each of the persons.
4. Next, since several people can be mentioned within individual documents, we can assume that people which are co-referenced within a document are collaborating on the topic being discussed within the document. Based on this we create a network of collaborations between the employees.

Based on the fact that collaborations and topics change through time, we are able to construct from the extracted pieces a **DynamicNetwork** representing a model of an enterprise. By using the **DynamicNetwork** we will be able to solve several other tasks which will be represented as specific insights into so constructed data structure. In particular, based on the above descriptions what can be extracted from the documents, the components of the **DynamicNetwork** can be defined as following:

- Each **Node** represents one person or group of people (organizational unit) being mentioned within the documents. Each **Node** has additional attached static data like meta-data from the database of employees and topics in which the person (or organisation) appears within the documents. Furthermore, since we deal with a timeline of documents, each **Node** can have topics organised in the temporal way enabling temporal topic analysis.
- Each **Link** connects two people (or a person with an organisational unit) which are co-referenced within the documents. A person can be linked also with an organisation based on the information from the database of employees. An additional information on the link are the topics of the documents where two entities get co-referenced which enables to infer on which topics two entities work together. Similarly as for the **Node**, the topics can be organised in the temporal way.

Such a **DynamicNetwork** enables to observe dynamics of an enterprise including:

- competence profiling of people and organisations,
- collaboration between people and organisations,
- detecting trends in collaboration and topics,
- delivery of contextual information (e.g. from the personal or organisational point of view),
- providing a ground for designing incentives,
- ...

6.2.3 BT scenario

At this stage there are several possible paths that the BT case study scenario might take and the detail are not clear yet to the degree which would allow specific definition of the task. The setting we have involves the users that are accessing several systems/tools and several information sources, work mainly remotely and in smaller teams. At this stage we can conclude for the tasks performed within BT to be similar to the ones within Accenture case study with a possible additional need for predicting the next system/tool to be requested by the user (and thus preparing in advance all the prerequisites for accessing that system/tool).

References

- Fayyad, U., Piatetski-Shapiro, G., Smith, P., and Uthurusamy R. (eds.) (1996) *Advances in Knowledge Discovery and Data Mining*. MIT Press, Cambridge, MA, 1996.
- Hand, D.J., Mannila, H., Smyth, P. (2001) *Principles of Data Mining (Adaptive Computation and Machine Learning)*, MIT Press.
- Witten, I.H., Frank, E., (1999) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann.
- Aggarwal, C., (2006) *Data Streams: Models and Algorithms (Advances in Database Systems)*
- Gama J., Gaber, M.M., (2007) *Learning from Data Streams: Processing Techniques in Sensor Networks*
- Lafferty, J.D., McCallum, A., Pereira, F.C.N. (2001) *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*, Proceedings of the Eighteenth International Conference on Machine Learning
- Cristianini, N., J Shawe-Taylor, J. (1999) *An introduction to Support Vector Machines: and other kernel-based learning*, Cambridge University Press
- Manning, C. Schütze, H. (1999) *Foundations of Statistical Natural Language Processing - MIT Press*. Cambridge, MA
- Mannila, H., Toivonen, H., Verkamo A. (1995) *Discovering frequent episodes in sequences - Proceedings of the First International Conference on KDD*