Specification of Transitions in CEP systems using Context Feature Models

Bachelor-Arbeit
Niels Danger
KOM-B-0609

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Specification of Transitions in CEP systems using Context Feature Models
Bachelor-Arbeit
KOM-B-0609

Eingereicht von Niels Danger
Tag der Einreichung: 15. April 2018

Gutachter: Prof. Dr.-Ing. Ralf Steinmetz
Betreuer: Manisha Luthra
Betreuer: Markus Weckesser

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation (KOM)
Prof. Dr.-Ing. Ralf Steinmetz

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor-Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die schriftliche Fassung stimmt mit der elektronischen Fassung überein.

Darmstadt, den 15. April 2018 Niels Danger

Contents

List of Figures

List of Tables

1 Introduction and Motivation

With the ever increasing number of hard- and software systems that generate continuous streams of heterogeneous data it becomes more difficult to extract meaningful information from this data and act upon it in real time. Complex Event Processing (CEP) is a method that allows processing data streams from multiple sources by letting users specify events of interest as continuous queries. Events resulting from the combination of several simple and other complex events through aggregation, composition and derivation operators are called complex events. If the CEP system detects a complex event that matches a query, all interested users will be notified of the event's occurrence immediately. If processing of the operators involved in detecting complex events is performed centrally on one computing node, scaling of the system in terms of joining data sources, additional queries or an increase of the data rate is obviously limited. Therefore, distributing the operators involved in the processing of the data over several computing nodes is beneficial, if not necessary, in many application scenarios.

Since applications that have to react in real time to occurring events require a certain Quality of Service (QoS) from the CEP engine, a solution to include QoS demands in a CEP system has been proposed with AdaptiveCEP [12]. It allows the specification of QoS demands on queries, e.g. that the end-to-end latency from data source over all participating processing nodes to the user has to be less than 100ms, or that the frequency of received events has to be above a certain rate. To satisfy such requirements, the choice of a fitting placement algorithm is crucial as it is responsible for the distribution of individual processing steps of a query onto a network of processing nodes. Every placement algorithm is designed with specific assumptions about processing network characteristics (e.g. mobility, density) and QoS metric optimization goals in mind. These characteristics and the QoS demands from users constitute the context the system is operating in. Because of this, employing the wrong placement algorithm in a given context can have serious negative impact on the relevant QoS metrics and degrade performance of an application relying on the CEP system.

1.1 Problem Statement

While in a static context only the initial choice of the placement algorithm is important, in an application scenario with dynamic context the optimal placement algorithm may change over time because the network conditions or QoS demands can change significantly, making a switch from one placement algorithm to another necessary. Such a change of mechanism is called a transition, triggered by the system moving from one context to another. The aspects that can be responsible for such a change are, among others, node topology (clustered vs. distributed) and traffic type (burst vs. uniform). The ability to dynamically reconfigure the mechanisms of the system at run-time based on the context is necessary to assure stable and reliable system operation in the face of significant changes in network conditions and user demands. In this work we will focus on reconfiguring the most influential mechanism, namely the deployed placement algorithm, but any other mechanism could be reconfigured as well.

In order to perform reconfigurations of the mechanisms at run-time, we need two pieces of information: first, up-to-date information on the current context to determine if a significant environment change has occurred and what the new context is; and second the most fitting mechanism(s) to transit to in the new context.

According to Dey [5], context is "*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application*". In our case, relevant information is any information that helps determining what mechanism to use (i.e. information about network conditions, network QoS metrics, user QoS demands) without the user having to set it manually. This information must be captured in a structured way so that it can be automatically processed and used to determine the

placement algorithm to be used. Due to the inherent variability of both system (utilized placement algorithm, processing nodes) and context we need a modeling technique that is suited to capture the possible configurations of our system and its context in a coherent way. Such a technique is available in the form of Dynamic Software Product Line (DSPL) which can be represented as graph-based Feature Model (FM). Prior work has shown their ability to model context-based adaptation of a system with the help of so called Context-aware Feature Model (CFM) [11] [1]. Feature Models can represent (Dynamic) Software Product Lines which model different variants of a software system. They are used to specify variation points and the possible alternatives (features) at these points as well as constraints between these alternatives. A product configuration is created by selecting or deselecting every feature, and is only valid if it respects all constraints between features.

As the several variable parts of the context are important for deciding what adaptation to make in the system, modeling them as separated System Features (directly configurable system parts, e.g. placement algorithm) and Context Features (influences that cannot be controlled directly, e.g. network metrics, QoS demands) together in one model is a reasonable choice. By interpreting different context states as different configurations, we can identify changes between contexts by observing changes in the configuration of the context side of the CFM and then, based on these observations, trigger appropriate transitions from one mechanism to another, represented by a reconfiguration of the system side of the CFM. By structuring the model through constraints and dependencies (between context features and system features as well as among context features) the space of possible configurations is limited to valid ones. This limits the amount of possible reconfigurations of the system, in our case the choice of the placement algorithm, to those allowed by the model. Our aim is to determine the most fitting placement algorithm for a given configuration of context features; to achieve this, we propose to either predict the performance of every placement algorithm that is allowed by the constraints in this context configuration through some form of regression, or train a classifier to predict the most fitting placement algorithm based on the current context. In the first case, model constraints limit the number of different placement algorithms to be considered for a given context; in the second case, the amount of required training data is limited to instances with valid context configurations.

Automating the choice of a fitting placement algorithm would enable the system to adapt to known situations as well as new ones without the need for manual reconfiguration of the employed placement algorithm.

1.2 Goals

This thesis aims to explore context-dependent transitions of mechanisms in CEP systems and how to model them in order to enable self-adaptation of the system's adaptation behaviour, specifically the placement algorithm. To achieve this, the following contributions will be made:

- Review of relevant literature on context modeling, especially with the help of feature models, and context-based adaptation of event-based systems
- Identification of required properties of a context model to represent an adaptive CEP system
- Design of a CFM that allows to capture context changes and QoS demands. This model allows to restrict the space of valid mechanism configurations eligible for a transition
- Design of a machine learning approach for placement algorithm selection based on user QoS demands and constraints imposed by the CFM
- To prove the applicability of our approach, the existing CEP system AdaptiveCEP will be extended with a prototypical implementation of the designed context model and a module that allows adapting the deployed placement algorithm at run-time based on the context information gathered by our model

- Evaluation of the implementation in selected simulated scenarios to quantify potential performance gains.

1.3 Outline

The remainder of this document is structured as follows: chapter 2 includes background information on the topics of Complex Event Processing, Context Feature Models and Machine Learning and surveys and analyzes related work in the area of Context Feature Models in Dynamic Software Product Lines and adaptive CEP systems. Furthermore we will introduce an application scenario to give a practical example to refer to in the next chapters. In chapter 3 we will perform a context model requirement analysis for AdaptiveCEP, present and justify our CFM design and explain the prototypical implementation of the model and the adaptation mechanism. Chapter 4 covers the evaluation methods, results and their discussion In the final chapter 5 we will recap the thesis and reflect on the proposed solution and its results as well as possible future work.

2 Background

2.1 Event and Data Stream Processing

The task of processing streams of data produced by multiple sources has been approached from two different research directions. The first approach is inspired from the area of Database Management System (DBMS) which persistently store data and allow users to query it on demand. As this is not an applicable approach when dealing with applications that need to process continuous streams of data from from many sources and update the results of queries made to them by consumers in real-time, Data Stream Management Systems (DSMSs) have been developed to address this issue. DSMS allow users to apply SQL-like queries to potentially unbounded streams of data in a vein similar to DBMS. However, a major difference arises from the continuous nature of processed data: while in traditional DBMS a finite amount of data is stored persistently and can be accessed anytime, data streams are potentially unbounded and not persistent. This entails that instead of applying an incoming query to existing data, incoming data is applied to an existing query. Queries are treated as continuous (or standing) and update their results based on incoming data. They consist of common SQL operators like aggregations, select, join and all the operators defined in genral by relational algebra [3]. However, since DSMS usually do not support an ordering of arrived data, the detection of patterns in the processed data through operators like sequence is not possible.

The second approach is called CEP and views data streams as streams of events since it was developed in the context of event-based simulation [9]. It can be seen as an extension to the content- and topic-based publish-subscribe paradigm [3], as it allows subscriptions to refer not only to single events, but to events created through the combination of lower-level events from multiple heterogeneous sources. An *event* can be defined as anything that happens [2], or more precisely, any meaningful change of state in the universe of an application that can be perceived through some kind of (physical or virtual) sensor. Depending on the application, a mere change in time can be an event, so any ordered sequence of data generated by a source like a light sensor can be interpreted as an *event stream*. To be precise, when we use the term *event* we actually mean a *notification about an event*, not the event itself that caused the change of state. This *event notification* is a message that contains one or more attribute-value fields and usually has a time stamp and a type.

CEP allows more involved analysis of event streams since a time-stamped event model allows the events to be ordered, enabling sequence operators and pattern matching. This in turn makes the detection of causal relationships possible (if in a domain A causes B, a situation that is characterized by the occurrence of event A followed by event B can be detected) The of goal CEP is to extract higher-level information from a stream of lower-level events by declaratively specifying events of interest through event patterns, often expressed in a query language. This allows the timely detection of events representing situations of interest that demand a reaction.

The processing of events with CEP can be summarized as follows: Data sources (producers) generate streams of simple events (e.g. sensor readings) which can be processed using operators that combine and correlate them with other events and domain knowledge to create complex events. Complex events in turn can serve as input for further operators. Interested parties (consumers) can subscribe to events of interest to them. These events of interest are represented by complex events and can be defined through queries consisting of aggregation, composition and derivation operators, which in turn are executed on event streams by the CEP system.

Queries - sometimes also referred to as event patterns [10] or event definition rules [4] - consist of one or more operators connected in a graph. CEP operators can be categorized as follows [6]:

- Aggregation: operations that are applied to a finite set of values, like count, sum, average, median, minimum, maximum. Since streams are potentially unbounded, some form of windowing (tumbling or sliding window of the last x events or of the last y seconds) has to be applied to the stream so it can be processed by these operators.
- Composition: combination of information from two or more events, not necessarily of the same type, into a new event. This event conveys new information by relating two or more events to each other
- Derivation: using knowledge about domain semantics and/or external knowledge, a new event is derived through reasoning about information from incoming events and information from knowledge

The event processing system is responsible for processing of incoming events and routing detected complex events to any interested consumers. This allows for timely reaction to new information by the consumers, enabling a range of real-time, reactive applications.

As an example, an event pattern correlates incoming 'Smoke Particle Detected' events from a smoke particle sensor with aggregated 'Temperature' events if the average temperature of the last 30 seconds exceeded 50 degrees Celsius. If the CEP system detects a series of events from sensors in close proximity to location X that matches this pattern, it creates a complex event 'Fire in location X detected' and emits it to the interested parties (e.g. building monitoring system).

CEP systems have various applications, among them the domains of intrusion detection systems, credit card fraud detection, stock market trading, business intelligence, network monitoring, traffic monitoring, healthcare, transportation, supply chains and manufacturing. Due to the nature of some of these domains, event sources can be numerous and widely distributed and producing high amounts of data, making CEP systems that rely on central processing susceptible to scaling issues. This calls for a distributed computing approach, which exists in the form of Distributed CEP.

While DSMS systems rely mostly on aggregation operations to perform bulk data processing, CEP systems traditionally focus more on detecting temporal and causal correlations in order to abstract to a higher level of information [3]. Since both can be operated on a network of processing nodes, both need operator placement algorithms (which will be introduced in the next section) that are part of the mechanisms whose transitions are a focus in this work.

2.1.1 Distributed Complex Event Processing

The operator graph of a query can either be centrally processed, or its processing can be distributed over an overlay network of several independent processing nodes (called hosts if an operator is deployed to them). Distributed processing makes CEP systems much more scalable: events do not incur additional producer-to-consumer latency from being routed to a central processing unit, network traffic is decreased because unnecessary events can be filtered on hosts closer to their sources, and operators can be replicated onto multiple hosts in case of high data rates and/or limited processing capabilities of hosts, allowing the system to scale in and out.

2.1.2 Adaptation mechanisms in CEP

%placement algorithms
%purpose, assumptions, optimization goals, central vs decentral, task assignment problem NP-hard
% TODO rework, taken from old intro
Placement algorithms are utilized to find and maintain a mapping of the queries submitted to a CEP system to the network of processing nodes which is optimal w.r.t. some non-functional property (such as latency, bandwidth, load or combinations thereof). Every placement algorithm attempts to optimize one or more non-functional properties and is designed for specific situations characterised by network

conditions like traffic type (bursty or uniform), network topology (distributed or clustered), and stability of connections and queries.

These algorithms handle the distribution of the individual operators onto the network of processing nodes. For this several pieces of information are needed [8]:

- the queries to be placed, viewed as logical flow graphs consisting of event sources, operators, and drains
- the network of interconnected processing nodes that will host the operators of the flow graph
- measures or estimates of non-functional properties of the network and its nodes that are decisive for the placement choices to be made, including, among others, latency and bandwidth of links, availability, processing speed of operators on a node
- information about processing nodes' capabilities and operators' requirements that may constrain the placement

The operator placement problem is a specific instance of the task assignment problem which has been shown to be NP-hard, hence placement algorithms use some heuristic function that is minimized/maximized to find a placement that is close to the optimal one.

Placement algorithms can work in a centralized or distributed fashion, meaning that they have either information on the state of the entire network, allowing them to find globally optimal placements, or are restricted to local information from the node they are executed on and its neighbours, allowing them to make locally optimal placement decisions. Centralized algorithms tend to find globally better placements, but are restricted in their scalability due to the amount and frequency of necessary information exchange.

Most mechanisms periodically re-evaluate the current placement to check if it is still optimal or needs to be adjusted. While periodic re-evaluation of the placement enables the system to react to changes in the underlying network to a certain degree, there is currently no way to automatically adapt the utilized placement algorithm to major changes in the environment (context) that adversely affect the QoS demands. If any part of the network or application context (e.g. network conditions, network topology, type of QoS demands on queries, event frequency) is changed significantly, the placement algorithm that was used before may no longer be optimal as it was intended for a different context. The choice of the most fitting placement algorithm for the current context is crucial for system performance.

The placement of the operators cannot happen in an arbitrary way: some operators require a minimum of processing power or specific hardware; it is further constrained by sequential dependencies between operators: if an operator combines several simple or complex events, the operators that process these events need to be placed on processing nodes located upstream (viewed from the drain) of this operator's processing node, so this can reduce the number of feasible solutions.

Further mechanisms to satisfy QoS demands involve the optimization of the operator graph as well as parallelization by replication of operators.

The choice of the appropriate strategy and its mechanisms depends on both the network conditions and the QoS demands imposed on the queries by users. As both can change over time, some form of context awareness and adaptability is necessary to guarantee a steady level of system performance.

2.1.3 Quality of Service in CEP

%TODO re-use and re-write partly if possible %==== from OLD QoS+placement introduction, related work(placement) ====
This becomes even more important in a scenario with geographically spread out producers and consumers where a certain end-to-end latency must not be exceeded. In this case it is not possible to route all events to a central processing node as it will likely cause the latency limit to be exceeded. Therefore the computations involved in processing a query need to be distributed across several nodes.

Placement algorithms handle the placement of the individual operators on hosts and try to optimize one or more QoS metrics in doing so and are periodically re-evaluated to check if the placement is still optimal or needs to be adjusted. These QoS metrics can be, among others:

- latency
- bandwidth consumption
- load balancing
- reliability
- availability
- energy consumption

Currently, most operator placement algorithms used in practice focus on optimizing latency and/or load balancing.

2.2 Dynamic Software Product Lines and Feature Models

% introduction with a small example
% terms and definitions, FODA notation
% relevance of context-awareness to enable adaptability of systems
% introduction to self-adaptive systems that profit from mechanism transitions, (examples in related work?)

2.2.1 Context-Aware Feature Models

% may be moved to related work as it is best explained in context of other work and is directly relevant to the thesis

2.3 Summary

3 Related Work

% relevant work on related topics
% categorize, evaluate, summarize other works, show differences and parallels; similar solutions but not necessarily similar problems; understand limitations of other works -> avoid these limitations if possible)

% This chapter should give a comprehensive overview on the related work done by other authors followed by an analysis why the existing related work is not capable of solving the problem described in the introduction. The chapter should have a length of about three to five pages!

3.1 Context Modeling for Event-based Systems

3.1.1 Context Modeling Techniques

% survey related work on techniques that don't use FM, point out pros and cons
% possible considerations for design and implementation, things to avoid and things to have

Freudenreich explicitly differentiates between situation context (information about environment) and interpretation context (information about data). Regarding the modeling of situational context of event-based systems he identifies the following required features and characteristics, based on three surveys of different context models:

- aspects:
  - absolute and relative space and time
  - application as subject
  - no context history or user profiles
- representation:
  - key-value based
  - low formality
  - fully general, variable granularity
  - no valid context constraints
- management:
  - context construction distributed and at runtime
  - basic context reasoning
  - hooks for incompleteness and context information quality monitoring
  - multi-context modeling

After identifying these requirements, he argues that there is a need for relationship representation and basic reasoning, but ontology-based models, which provide elaborate context reasoning, are considered to be too detrimental to EBS performance. Therefore he proposes an ontology-based metamodel that can represent domain conCEPts in terms of conCEPts, attributes and relationships which allows for context construction to happen at runtime, while the context information structure is determined at design time. The metamodel makes no assumptions about the domain model's data types or structure, making it very flexible. Situations and reactions to them can be specified declaratively as policies using the defined ontology.

E.Barrenchea et al. propose to embed context information directly into events as a first class element. Components can specify the context of their publications and/or subscriptions. Components can be grouped into contextual environments (making them "siblings") that share the same context parameters and values. Context awareness is realized by context filters in an environment middleware layer that filter according to the specified event schema and the environment they are part of, which relieves components of keeping track of context information directly. This moves context away from the component logic and towards the layer of the pub/sub scheme. An environment manager handles context updates and provides an interface to the application layer for utilizing context information.

Koripipää et al. present a software framework for context acquisition and processing in mobile scenarios. The API uses an extensible context ontology to define usable contexts. The context schema is defined in RDF syntax to ease the sharing of the ontology. Context has six possible properties: (first two are mandatory) 1. type 2. value 3. confidence 4. source 5. timestamp 6. attributes. It is possible to create context hierarchies and composite contexts. The framework consists of a context manager (blackboard, delivers context information based on subscriptions), a resource server (collects and preprocesses context data from sources) and context recognition services (create higher-level contexts from atoms, can be plugged in at runtime). The recognition services make use of Naive Bayes classifiers and the confidence property to create higher-level contexts.

3.1.2 Feature Models for Context-Aware Systems

% more detailed survey of Saller, Acher, Weckesser etc papers

Some authors propose using feature models, which have their origins in the domain of highly configurable software product lines (SPL), as a means to model the variability and behaviour of context-aware systems.

M.Archer et al. suggest using two separate feature models: one for the system and its variable parts, and one for the context, each of them with its own constraints (e.g. require, exclude). This allows for a homogeneous representation of system and context and the relationships between the two. The two models are then linked together by dependency constraints which represent the adaptation behaviour. The adaptation of the system happens based on event-condition-action (ECA) rules, with elements of the context feature model as condition and a re-configuration of the system feature model as the action part. The authors mention optimization of a utility function as an alternative to ECA-rules for modeling adaptive behaviour, which would require the feature model to be extended with attributes for information needed for the optimization.

Saller et al. aim to enrich feature models with context information and context constraints in order to limit the set of valid configurations further. This is done to limit the complexity of re-configuration planning and enable such calculations on resource-constrained devices. Reconfiguration is modeled and pre-planned in the form of a transition system, with transitions from one valid system configuration into the next, based on context. To further reduce the amount of possible transitions, states with different configurations that satisfy the same constraints are considered to be identical in terms of transitions (incomplete state space). This model allows for multiple contexts to be active at once.

Weckesser et al. propose an approach that extends the expressiveness of feature models as well as the possibilities to validate them. It allows for modeling of feature attribute types and constraints beyond boolean as well as UML-like multiplicities of features. Furthermore, they enable automated validation of such cardinality-based feature models through methods that check the constraints imposed on the feature model by cardinality annotations. This is useful when dealing with systems that have multiple instances of a feature and need to take dependencies among them (and their number) and other features into account.

3.2 Adaptation and Context-Awareness for EBS

% AdaptiveCEP paper

% Lakshmanan placement algorithm classification
The work of G. Lakshmanan et al. [7] is a survey of 8 algorithms developed to solve the problem of optimal operator placement in data stream processing systems. To compare the different approaches, the authors define a set of core components that significantly affect the performance of the system and compare the categories of these components:

- architecture (centralized, decentralized or hybrid implementation of placement logic)
- algorithm structure (centralized vs decentralized decisions)
- metric(s) to optimize (load, latency, bandwidth, machine resources, operator importance, combinations of those)
- operator-level operations (reuse or replication of operators)
- reconfiguration (triggers for operator migration: thresholds, periodic re-evaluation, constraint violation)

Based on these components, 8 placement algorithms developed rEvent Stream Processing (ESP)ectively by Pietzuch, Balazinska, Abadi, Zhou, Kumar, Ahmad, Amini and Pandit are assigned to the discussed categories. The authors then proceed by comparing the algorithms w.r.t. their design decisions and underlying assumptions. It is concluded that decentralized approaches which are able to adapt dynamically by operator migration are prevalent, with latency and, by extension, load balancing being the preferred metrics to optimize. Based on the comparison of the assumptions a decision tree for selecting a fitting placement algorithm considering the characteristics of a given problem (node distribution, number of administrative domains, dynamics of topology and data, query diversity) is proposed.

3.3 Performance Prediction using Feature Models

% survey of SPLConqueror (if not used, replace section heading in accordance with other ML-related papers)

SPLConqueror is a collection of programs and a library for discovering the influence of configuration options of software with variable elements on non-functional properties. This is done iteratively by using linear regression learning and stepwise feature selection. It consists of four programs:

- VariabilityModelGUI: creation and editing of variability models of the configurable system; allows for exclude/require constraints between elements
- PerformanCEPredictionGUI: learning an influence model from a given variability model and a set of measurements (of configurations and their performance regarding one metric). Possible configuration options for sampling and linear regression algorithm are available. The result is a an influence model of terms of weighted influences on one specific performance metric in the following form: (w1*c1 + w2*c2 + ... w34*c3*c4) where w34*c3*c4 denotes the selection of both configuration options c3 and c4
- SPLConquerorGUI: GUI for visualization and analysis of the learned model with options for filtering variables and adjustment of the model. Visualization of influences and detected dependencies
- CommandLine: allows for specification and execution of experimental setups with different sampling and machine learning options

Further submodules include MachineLearning (algorithm for learning performance-influence model; interface for SAT checking of configurations w.r.t variability model and optimization of configuration for a

given non-functional property and objective function) and PyML (interface to scikitlearn (a python ML framework), different regression learning techniques).

Application to AdaptiveCEP: 1. determine from measurements for every mechanism (context (network conditions) -> performance (w.r.t. mechanism's optimized QoS metric?)) the influence of every context element on the performance metric by using linear regression; -> need to restrict combinations of context features + mechanisms to valid ones to limit necessary training/measurement data 2. the resulting formula (with weights for the influence of each context feature) can predict the performance of every context feature configuration for each mechanism 3. with this, calculate the predicted performance of every mechanism for a given context configuration 4. compare the performance predictions of the different mechanisms for a given context configuration; problem: how to compare performance measured in different metrics? Can it be reasonably normalized to [0,1], if yes, what lower/upper bounds to choose for latency, bandwidth, ... to make them comparable? 5. if reasonable comparison of performance metrics can be found: choose mechanism with best (normalised) performance for the given context

3.4 Comparative Analysis and Discussion

3.5 Summary

4 Context-aware Feature Model and Mechanism Transition Approach

% This chapter should describe the design of the own approach on a conCEPtional level without mentioning the implementation details. The section should have a length of about five pages

% in detail: problem analysis, identification of important issues, describe and explain proposed solution, identify remaining issues

4.1 Context Feature Model Design

4.1.1 Requirement Analysis

In the case of AdaptiveCEP, the system should be able to detect the conditions of the network that hosts the operators of its queries. In order to ensure the compliance with any QoS demands placed on the queries, it should then be able to dynamically choose appropriate mechanisms (operator placement strategy, ...) that fulfill the demands and optimize their relevant QoS metrics even if the network conditions change. To do this, the application needs knowledge about the status of any nodes currently hosting operators (to detect performance deterioration) as well as alternative nodes currently unoccupied (for possible reconfiguration). Some of this information could be abstracted into coarser categories to facilitate decision making. Furthermore, it should be able to cope with joining and leaving hosts. The adaptability of the application is given by the different strategies therefore it should be possible to include new strategies, possibly with new types of QoS metrics. Useful dimensions of network status information are, among others: latency, bandwidth, utilization, throughput, location/velocity (in case of mobile network entities), energy consumption.

To capture the requirements of the context model in a structured manner, the analysis framework proposed by Bolchini et al. is used. This categorizes the requirements as follows:
1. Modeled Aspects

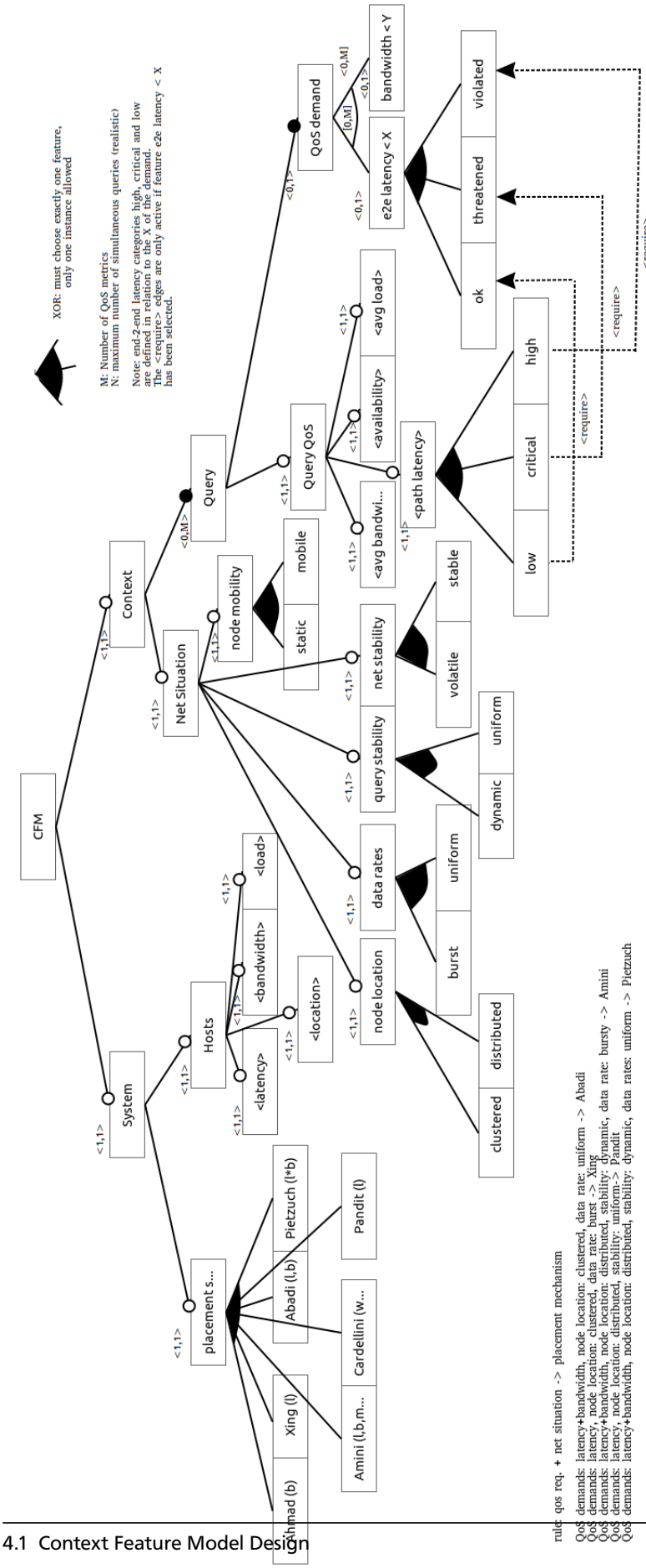
Feature	**Analysis Result**
space	yes, necessary for proximity demands
time	maybe, for "freshness" of information
absolute/relative space and time (coordinates/timestamp vs. near/after something)	abs.+relative space, absolute time
context history (does the current context state depend on previous ones)	unclear, maybe for stability metrics of links; needed for machine learning purposes?
subject (what is the point of view -> user or app perspective)	app centric, system needs context information to optimize strategy usage
user profiles (are user preferences relevant for the context, how are they represented)	QoS demands can be viewed as user preferences

2. Representation Features

Feature	**Analysis Result**
type of formalism (key-value, markup, logic, graph, ontology -> different intuitiveness, reasoning possibilities)	Context Feature Model == graph-based, logic for constraints
flexibility (domain specific vs general)	specific to application, with enough flexibility for extensions
variable context granularity (ability to represent context at different levels of detail)	-
valid context constraints (can the number of possible contexts be restricted by semantic constraints)	necessary to restrict space of system configurations; inherent to CFM

3. Context Management and Usage

Feature	**Analysis Result**
context construction (central description of possible contexts at design time vs distributed construction of the current context at runtime)	central description of context dimensions, distributed construction by collection of current values
context reasoning (ability to infer properties or construct higher-order context information from sensor readings)	maybe, abstract from measurements to situations
multi-context modeling (one instance of the model can represent all possible context, not one instance for each context)	yes, need to model changes between contexts


CFM
System
Context
placement s...
Hosts
Net Situation
Query
<latency>
<bandwidth>
<load>
<location>
node location
data rates
query stability
net stability
node mobility
Query QoS
QoS demand
<avg bandwi...
<availability>
<avg load>
<path latency>
e2e latency < X
bandwidth < Y
clustered
distributed
burst
uniform
dynamic
uniform
volatile
stable
static
mobile
low
critical
high
ok
threatened
violated
Ahmad (b)
Xing (l)
Amini (l,b,m...
Cardellini (w...
Abadi (l,b)
Pietzuch (l*b)
Pandit (l)
XOR: must choose exactly one feature, only one instance allowed
M: Number of QoS metrics
N: maximum number of simultaneous queries (realistic)
Note: end-2-end latency categories high, critical and low are defined in relation to the X of the demand.
The <require> edges are only active if feature e2e latency < X has been selected.
<require>
<require>
<require>
rule: qos req. + net situation -> placement mechanism
QoS demands: latency+bandwidth, node location: clustered, data rate: uniform -> Abadi
QoS demands: latency, node location: clustered, data rate: burst -> Xing
QoS demands: latency+bandwidth, node location: distributed, stability: dynamic, data rate: bursty -> Amini
QoS demands: latency, node location: distributed, stability: uniform-> Pandit
QoS demands: latency+bandwidth, node location: distributed, stability: dynamic, data rates: uniform -> Pietzuch

Note: will split graph in system and context part for better readability

The model limits itself to those elements of the system that are relevant for the detection of or reaction to network environment changes. The graph modeling technique used to represent the context model as a feature model was first used in the domain of Software Product Lines (SPL) and allows definition of variable parts (features) of both system and context. At runtime, the active features constitute the configuration of the system/context; the space of possible configurations is restricted by constraints between features (xor-, or groups, require- and exclude relations). This allows the restriction of the model to valid configurations, which is helpful when gathering measurement data for different possible configurations. The main elements of the context are the queries executed by the system, their QoS metrics, the QoS demands placed upon them, and a set of situations that characterize the network conditions of the nodes in the system. These situations abstract from measurements on single hosts to the overall state of the queries' environment, and can be used in the definition of constraints and adaptation rules. Their mutually exclusive subcategories (like dynamic, uniform) can be defined by threshold values (in some relation to QoS demands on operators) and can be defined more finely granular if necessary. Note that multiple Situations can be active at the same time. The variable parts of the System are the ones that can be directly reconfigured: the placement mechanisms, each optimizing a set of QoS metrics and suited to a certain situation. Although they can not be directly reconfigured, the Hosts/Nodes are an integral part of the system, so they are included on this side of the model. Every Host in the System has a set of associated QoS measurements. For the sake of readability, constraints among the features of the model can as well be noted separately as follows:

(C1) Situation:data rate:burst **and** Situation:node location:clustered **and** Query:QoS demand: e2e latency < X **requires** placement:Xing

with Xing being a placement strategy that aims to minimize the latency. Adaptations from one strategy to another could be triggered in a similar manner:

(A1) Query:QoS demand:latency **and** Query QoS:path latency:high **implies** other latency optimizing strategy

4.2 Mechanism Transition Prediction Approach

4.3 Summary

5 Implementation

% This chapter should describe the details of the implementation addressing the following questions: %1. What are the design decisions made? %2. What is the environment the approach is developed in? %3. How are components mapped to classes of the source code? %4. How do the components interact with each other? %5. What are limitations of the implementation? %The section should have a length of about five pages.

Preliminary Implementation Considerations:

- context information could be distributed via context events; a context manager subscribes to context sources and the application (or adaptation strategy) subscribes to the aggregated events from the context manager
- context source monitoring could happen similar to the existing MonitorFactories
- create a new interface for the context elements (metrics, attributes, constraints) and its management

5.1 Design Decisions

5.2 Architecture

5.2.1 Context Feature Model

5.2.2 Mechanism Transition Prediction Approach

5.3 Interaction of Components

5.4 Summary

%This chapter should describe how the evaluation of the implemented mechanism was done. %1. Which evaluation method is used and why? Simulations, prototype? %2. What is the goal of the evaluation? Comparison? Proof of conCEPt? %3. Wich metrics are used for characterizing the performance, costs, fairness, and efficiency of the system? %4. What are the parameter settings used in the evaluation and why? If possible always justify why a certain threshold has been chose for a particular parameter. %5. What is the outcome of the evaluation? %The section should have a length of about five to ten pages

6 Evaluation

6.1 Methodology

% describe how performance is measured

6.2 Evaluation Setup

% describe testenvironment setup % describe test cases for transitions between mechanisms triggered by context changes

6.3 Results

% compare system performance in test cases with and without context awareness (if there are implementation variants, compare these as well)

6.4 Analysis and Discussion

% does the solution improve performance, what issues are there and why

7 Conclusion

%This chapter should summarize the thesis and describe the main contributions of the thesis. %Subsequently, it should describe possible future work in the context of the thesis. What are limitations of the developed solutions? Which things can be improved? The section should have a length of about three pages. % brief recap of work so far, discuss findings of analysis -> what are strengths and shortcomings of the solution, new problems -> future work

7.1 Summary

7.2 Contributions

7.3 Future Work and Final Remarks

Glossary

CEP Complex Event Processing
CFM Context-aware Feature Model

DBMS Database Management System
DSMS Data Stream Management System
DSPL Dynamic Software Product Line

ESP Event Stream Processing

FM Feature Model

QoS Quality of Service

Bibliography

[1] Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, and Jean Paul Rigault. Modeling context and dynamic adaptations with feature models. *CEUR Workshop Proceedings*, 509:89–98, 2009. ISSN 16130073.

[2] Mani K Chandy, Opher Etzion, and Rainer Von Ammon. The event processing manifesto. *Event Processing*, (10201):1–60, 2011. ISSN 18624405. URL http://drops.dagstuhl.de/opus/volltexte/2011/2985.

[3] Gianpaolo Cugola and Alessandro Margara. Processing flows of information. *ACM Computing Surveys*, 44(3):1–62, 2012. ISSN 03600300. doi: 10.1145/2187671.2187677. URL http://dl.acm.org/citation.cfm?doid=2187671.2187677.

[4] Gianpaolo Cugola and Alessandro Margara. Deployment strategies for distributed complex event processing. *Computing*, 95(2):129–156, 2013. ISSN 0010485X. doi: 10.1007/s00607-012-0217-9.

[5] Anind K Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001. ISSN 16174909. doi: 10.1007/s007790170019.

[6] Tobias Freudenreich. Simplifying the use of event-based systems with context mediation and declarative descriptions. 2015.

[7] Geetika T. Lakshmanan, Ying Li, and Rob Strom. Placement strategies for internet-scale data stream systems. *IEEE Internet Computing*, 12(6):50–60, 2008. ISSN 10897801. doi: 10.1109/MIC.2008.129.

[8] Geetika T. Lakshmanan, Ying Li, and Rob Strom. Placement strategies for internet-scale data stream systems. *IEEE Internet Computing*, 12(6):50–60, 2008. ISSN 10897801. doi: 10.1109/MIC.2008.129.

[9] David Luckham. What's the Difference Between ESP and CEP?, 2006. URL http://www.complexevents.com/2006/08/01/what's-the-difference-between-esp-and-cep/.

[10] David Luckham and Roy Schulte. Event Processing Glossary, 2011. URL http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2/.

[11] Karsten Saller, Malte Lochau, and Ingo Reimund. Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. *Proceedings of the 17th International Software Product Line Conference co-located workshops*, pages 106–113, 2013. doi: 10.1145/2499777.2500716. URL http://dl.acm.org/citation.cfm?id=2500716.

[12] Pascal Weisenburger, Manisha Luthra, Boris Koldehofe, and Guido Salvaneschi. Quality-Aware Runtime Adaptation in Complex Event Processing. *Proceedings - 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2017*, pages 140–151, 2017. doi: 10.1109/SEAMS.2017.10.