A Transition-aware Evaluation Framework for Adaptations in CEP Mechanisms

Bachelor-Arbeit
Niels Danger
KOM-type-number

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

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

A Transition-aware Evaluation Framework for Adaptations in CEP Mechanisms
Bachelor-Arbeit
KOM-type-number

Eingereicht von Niels Danger
Tag der Einreichung: 18. Oktober 2017

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

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 18. Oktober 2017 Niels Danger

Contents

Abstract

1 Introduction

1.1 Problem Description

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 in real time. Every data item that is generated can be viewed as an event, in the sense that two identical data items created at different points in time are two distinct events. This is the application domain of Complex Event Processing (CEP): Consumers can define queries as complex events that represent situations (or aspects of them) and event patterns that are of interest to them. Producers generate streams of data in the form of simple events (e.g. sensor readings) which can be aggregated and combined with other events (simple or complex) and domain knowledge to create complex events. If relevant events are detected in the event stream, the CEP engine will notify the interested consumer(s) immediately. This allows for timely reaction to the new situation by the consumers.

If the processing of the events happens in a centralized fashion, scaling of the system (in terms of joining producers, consumers or an increase of the event rate) is obviously limited. Therefore, distributing the processing of the data over several computing nodes is often beneficial, if not necessary. This becomes even more relevant if the producers and consumers are geographically spread out and short latencies from producers and consumers are important. The general problem can be described as follows: the query, in the form of a logical flow graph consisting of sources, operators and drains is to be mapped onto an underlying network of processing nodes that is connected to the producers and consumers. The sources and drains are pinned to the producers and consumers, so the operators are the only moveable part of the graph. This gives rise to the question as to how to optimally place the individual operators involved in satisfying the consumers' queries. The notion of what an optimal placement is can vary substantially depending on the application scenario and its specific Quality of Service (QoS) requirements. These QoS metrics can be, among others:

- latency
- bandwidth utilization
- load balance
- cost of transport and processing
- reliability
- availability
- energy consumption

The deployment of the operators to the nodes has to happen in such a manner that the most important QoS metric(s) are optimized. However, the placement of the operators cannot happen in an arbitrary way: some operators require a minimum of processing power or specific hardware, so this can reduce the number of feasible solutions. This 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.

Currently, most operator placement algorithms focus on optimizing latency and/or load balancing. While most of them are capable of reacting to changes in the CEP system and the underlying network to a certain degree, there is currently no way to adapt the utilized placement algorithm to major changes of the application environment that affect the QoS requirements. If the consumers' preferences of QoS metrics is changed, the placement algorithm that was used before may no longer be optimal as it was designed with a different goal in mind. Such a change of preferences can be caused by a change of

application context and could be detected by evaluating contextual information. This makes the ability to adapt the deployed operator placement algorithm beneficial for reacting to changes in the desired properties of the system.

1.2 Goals

The goals of this thesis are as follows:

- survey different QoS metrics and consumer quality preferences
- identify different CEP mechanisms that are influenced by context environment changes and changes in consumer preferences
- design and implement a transition-aware framework where multiple CEP mechanisms can be executed and adapted at runtime

2 Paper Summary

2.1 Operator Placement

The paper Placement Strategies for Internet-Scale Data Stream Systems by Geetika T. Lakshmanan, Ying Li and Rob Strom 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 respectively 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.

2.2 Context Modelling

Freudenreich explicitly differentiates between situation context (information about environment) and interpretation context (information about data). Regarding the modelling 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 modelling

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, he considers 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.

KoripipÃď'Ãď 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.

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 modelling adaptive behaviour, which would require the feature model to be extended with attributes for information needed for the optimization.

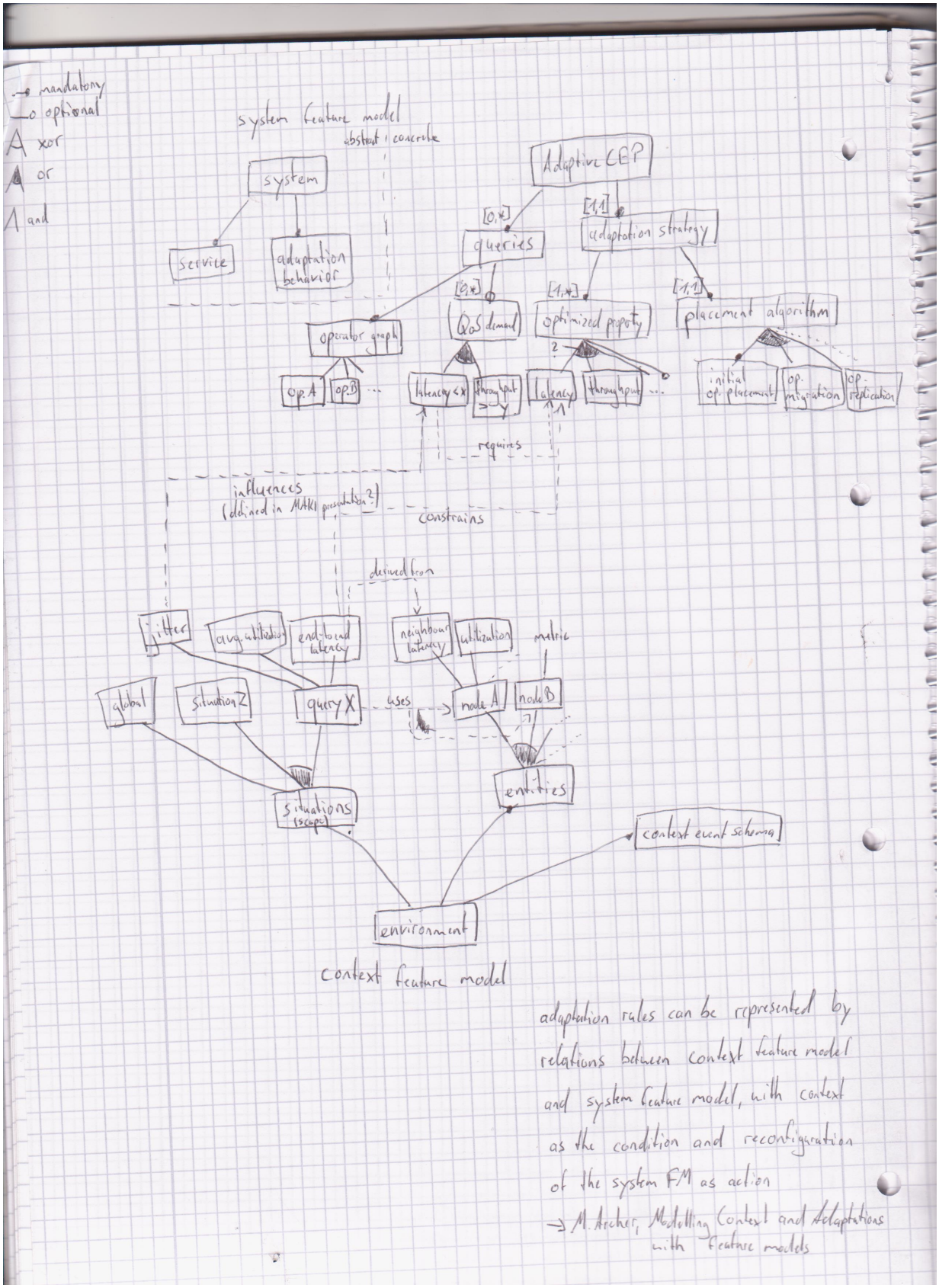
Saller et al. aim to enrich feature models with context information and context constraints in order to limit the set of valid configurations further. This is done to limit the complexity of re-configuration planning and enable such calculations on resource-constrained devices. Reconfiguration is modelled 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 as one 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 modelling 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 other features into account.

3 Context Model

3.1 Initial Draft



model.jpg

- context information should 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 ontology (components, attributes, relations) and its management
- implement the strategy interface described in the AdaptiveCEP paper to allow for specification of context adaptation behavior