Event Notification Services: Analysis and Transformation of Profile Definition Languages

Doris Jung and Annika Hinze

Department of Computer Science,
University of Waikato, New Zealand
{d.jung, a.hinze}@cs.waikato.ac.nz

Abstract. The integration of event information from diverse event notification sources is, as with meta-searching over heterogeneous search engines, a challenging task. Due to the complexity of profile definition languages, known solutions for heterogeneous searching cannot be applied for event notification. In this technical report, we propose transformation rules for profile rewriting. We transform each profile defined at a meta-service into a profile expressed in the language of each event notification source. Due to unavoidable asymmetry in the semantics of different languages, some superfluous information may be delivered to the meta-service. These notifications are then post-processed to reduce the number of spurious messages. We present a survey and classification of profile definition languages for event notification, which serves as basis for the transformation rules. The proposed rules are implemented in a prototype transformation module for a Meta-Service for event notification.

Table of Contents

1 Introduction to Event Notification Services

Event Notification Services are systems that inform subscribers of an event notification service about certain facts they are interested in (cf. Fig. 1). The knowledge about these facts is provided by publishers, which send information about these facts. In the following these are referred to as events e_x, to the event notification service. Events are changes of the state of an object such as a sensor. The interests of subscribers are defined in profiles p_x and registered with the event notification service. Whenever the event notification service receives an event which can be matched by one or several profiles, the respective subscribers are notified by a notification n_x [25].

Event notification services find their application in many areas such as medicine, logistics, quality and product management, the stock market, digital libraries or in traffic systems. For the area of healthcare many scenarios could be realised by using an event notification service such as depicted in Fig. 1.

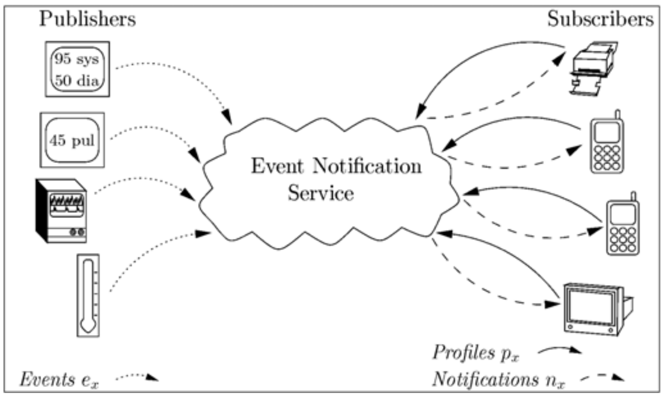

Fig. 1. Overview of an event notification service

Scenario 1 Notify Dr. Smith in case patient Anna Roberts develops an abnormally low blood pressure. Send an SMS to Dr. Smith's mobile phone in order to do so. Also, for future reference send this data to Ms. Roberts' electronic patient record.

Scenario 2 If John Donovan's pulse falls below 45, print out his heart rate at the central printer.

Scenario 3 If the electrocardiogram is malfunctioning notify the technician via an SMS to his mobile phone. Additionally, send a notification to request checking the costs of the reparation to the billing department using an automated e-mail.

We can express these scenarios with the help of the profiles and notifications given in Table 1.

Table 1. Profiles and notifications

Scenario	**Profiles p_x**	**Notifications n_x**
1	(p_1: ((sys < 95) OR (dia < 50)))	(n_1: (mpSmith: AnnaRoberts(sys/dia)) AND (recAnnaRoberts: AnnaRoberts (sys/dia)))
2	(p_2: (pul < 45)	(n_2: printer: hrJohnDonovan)
3	(p_3: (¬eECG)	(n_3: (mptechnichian: ECG broken) AND (emailbillingdepartment: check repair costs ECG))

1.1 Focus and Goals

The aim of this work is to examine how systems with differing profile definition languages can communicate with each other and to develop a concept for communication between systems on the ground of this analysis. The idea is to translate the semantics of all languages into a single collective language. This language should at least be able to map the power of each single profile definition language in order to avoid loss of information. When transforming those profile definition languages into a collective language, a problem occurs if differing languages have a different expressiveness and follow differing concepts. These differences have to be taken into account for the transformation of each profile definition language into the collective language. Otherwise, as a consequence there will be loss of information:

If for example a publisher does not support timeframes and we have a profile of a conjunction $(e_1, e_2)_{7sec}$, we will lose the information that these two events originally would have to appear within seven seconds, once we transform the profile to (e_1, e_2) without a timeframe.

To analyse the semantics of various profile definition languages, we research literature. This research forms the basis for a classification of the characteristics of the profile definition languages on account of their underlying concepts. Based on this classification we will subsume those profile definition languages into groups which offer a correspondence in concepts. This is necessary in order not to have to develop transformations for every single profile definition language. Also our approach causes an independence of the used system and thereby the possibility to permanently add new systems to the communication process.

For the transformation, identical operators of each group of profile definition languages, are written in a common meta-language and then transformed into the event algebra developed by Hinze and Voisard [26].

At the institute of computer science at Freie Universität Berlin two event notification systems have been under development, PrimAS [4] and CompAS [29]. PrimAS is based on primitive and CompAS on composite events. For better scalability, CompAS has been extended to a distributed system [5]. For this system, we have developed a prototype of a transformator [28], which translates the concepts of each group of profile definition languages into the event algebra on the basis of the developed transformations. This prototype is the basis for a transformator that can be integrated into the distributed version of CompAS representing a Meta event notification service (Meta-ENS). This Meta-ENS would be able to communicate with other event notification systems and furthermore to mediate between other event notification services.

1.2 Related Work

Language transformations In [11] Chang, Garcia-Molina and Paepcke have described an approach of how it is possible to offer a uniform query language for Boolean queries to users who are using different information systems. Users are provided with an interface with the help of which they can pass their queries. It is more powerful than the underlying systems. The implementation translates queries into the language of the target systems. Some of the languages of target systems are weaker; therefore the result set has to undergo post-filtering to gain the data which was originally looked for. The authors have developed three kinds of transformations in order to change languages into each other. These are useful for our work; nevertheless, they are not sufficient. Boolean queries are performed once only, however, event notification services have profiles which are active over a long history of time. Due to this, it is impossible to undertake post-filtering once and for all. Instead, we require a concept which permanently evens out the different degrees in power of various profile definition languages.

Related concepts There are some publications that deal with the analysis of profile definition languages such as [42] and [7]. Nevertheless, these are only by-products of analyses targeting different problems than this work. Therefore, they only marginally cover the presentation of profile definition languages which is relevant to this work. The analyses are undertaken with a small number of languages only and therefore they are not representative for the purpose of our work. Hence, they cannot be used in order to develop transformations between languages of event notification services. For the analysis of differing profile definition languages we cannot only use languages of event notification services but also those of several other systems. Next, we will describe what kind of systems may be considered for our approach. In doing so, we follow the account of Liebig and Buchmann [31] as well as Hinze [23]:

Active databases Active databases are database systems which have integrated certain rules into their database management system such as “on event do action”. There is no transaction concept. Furthermore, actions cannot be parameterised. A working assumption made is an event history which is totally ordered. Meanwhile, in a number of database management systems, simple rules are realised as triggers, e.g. Oracle and Sybase possess these kinds of mechanisms. Active database systems such as SAMOS [15] and Sentinel [9] (language: Snoop [10]) offer the concept of composite events.

Event-based infrastructures Event-based infrastructures are systems that support asynchronous message exchange in a distributed environment. Mostly, they are middleware-systems which realise communication via point-to-point connections. One application sends a message to a so-called event-channel. There the consuming application pulls the message. The Corba Notification Service [1] is an example for an event-based infrastructure.

Event-action systems A typical feature of event-action systems is that events which occur within the boundaries of the system can cause actions. Both events as well as actions are specified by the user. Actions can cause more actions in sequence. Specifications of these principles are also called ECA-rules (Event-Condition-Action Rules). Examples of this kind of system are the therapy planning programme PLAN [43] as well as YEAST [30]. Both are described in more detail in Section 3.

Integration into the MediAS Project This technical report is a summary of the thesis work fully presented in [28] (in German). That thesis is part of the project MediAS [24] of the group for databases and information systems at the Institut für Informatik at Freie Universität Berlin. This project dealt with the development of an event notification system which supports primitive events (version PrimAS [4]) and composite events (version CompAS [29]). It runs in a distributed environment (version DAS [5]). The work described in this technical report lays the foundation for the use of various event notification systems with differing profile definition languages as can be integrated into the distributed version of our system.

1.3 Structure

This technical report is structured as follows: In Section 2, we begin with an introduction of the concepts we are using within this work.

This is followed in Section 3 with the description of a survey of a number of profile definition languages and the systems that deploy them. For each system, we list the supported operators, support of time frames, consumption mode and duplicate handling. These analyses are based on the available literature, i.e. we refer to the operators and their parameters the way the initial publication does. Consequently, there are differences in the semantics and symbols compared to the ones we will introduce in Section 2.

Section 4 presents a comparative study of the profile definition languages we have analysed. The filter operators are translated into the terminology used in this technical report, i.e. here the descriptions of the systems described in Section 3 are explained using the same terminology for each system. Based on this, we perform a comparison of the approaches, and we group languages with similar concepts into categories (language groups). Following to that, the definitions of the resulting language groups are presented in the same section: Based on the comparative study, we identify five groups (types) of filter languages for event-based services. These language groups form the basis for the design of a meta-service for event notification and event-based communication.

Section 5 describes transformations we have developed to transform the language and its concepts of one profile definition language into another one. This is the precondition for the Meta-ENS mentioned above.

We have developed an implementation of our transformation rules as a proof of concept. This implementation, our transformator which is the prototype of our meta-service, is described in Section 6.

To finalise our technical report, in Section 7 we give a conclusion and outlook, evaluating our approach and discussing possibilities of future work.

2 Concepts

In this section, we introduce the concepts we are using in this technical report. Most of our definitions follow either [28], [5] or [27] which are based on the model for event notification proposed in [25]. A more extensive discussion of models and terminology can be found in [23].

Event Notification Service An event notification service is a system that sends subscribers (users or other services) a notification whenever an event is encountered this subscriber is interested in. Such an event is seen as change in state of an object as for example a wind sensor or a document. It is provided by publishers to the event notification service. There are two possible ways for the event notification service to discover these events: either they are actively collected (pull) or passively received by being sent to the event notification service by publishers (push). Publisher can be applications as well as other event notification services.

Profiles In order to describe relevant data, the subscriber has to register a profile, which he specifies in a *profile definition language* also referred to as a *filter language* (e.g. XMLCL [38] or CQ [32]). This way the profile expresses the interests of users. The profile consists of two parts the *query-profile* and the *meta-profile*. The former is described by attribute-value pairs, i.e. a description of the state of the object in relation to time. It is possible to connect these attribute-value pairs with the help of Boolean operators. Furthermore, the query-profile may have a timeframe, which denotes the time interval the event has to occur in. The latter contains information about the modes of notification such as the frequency of notifications and the format used for them.

Events An event describes the occurrence of a state transition of an object of interest at a certain point in time. Events are reported by means of notifications. Every event has a timestamp which indicates the time of its occurrence. Liebig and others have described in [31] that the evaluation of such timestamps can be problematic since publishers might originate from a distributed environment. But in such an environment there is no universal time. Nevertheless, in this work we abstract from these problems and undertake our analysis on a more theoretical foundation.

Event type Every event belongs to an event type *T*, which is defined by a set of attributes: $T = \{a_1, \ldots , a_n\}$. This way, *T* describes the structure of an event. Each attribute has a domain *D*. *D(a)* denotes the Domain of each attribute *a*.

Event instance In the following, we differentiate between event type and event instance. An event instance belongs to an event type exactly if it contains all the attributes of the event type.

Event class We distinguish event instances from event classes. An event class is a set of events specified by a profile while an event instance relates to the actual occurrence of an event. In the following, we simply use the term event whenever the distinction is clear from the context. Events (instances) are denoted by lower Latin *e* with indices, e.g. e_1, e_2, while event classes are denoted by upper Latin *E* with indices, e.g. E_1, E_2. The fact that an event e_i is an instance of an event class E_j is denoted as membership, e.g. $e_i \in E_j$. This relationship is non-exclusive, i.e. $e_i \in E_j$ and $e_i \in E_k$ is possible even with $E_j \neq E_k$. Event classes may also have subclasses, so that $e_i \in E_j \subset E_k$. The timestamp of an event $e \in E_1$ is denoted as *t(e)*.

Primitive vs. composite events There are primitive as well as composite events. Primitive events describe a change of state for only one object while composite events may refer to a change of state for several objects. They are nested and build up from primitive and other composite events. Within these composite events there may exist temporal dependencies such as present in the definition of a sequence of events. For this, we need to assume a temporal order of the events. A further dependency is the dependence on certain points in time or time intervals. This can be expressed by the means of an event algebra.

Hierarchical event notification services Hierarchal event notification services are able to use other event notification services as publishers. Thereby own events and events of other systems may be nested into composite events and be evaluated as such. This way, systems become more scalable and are able to offer more information to their users.

Event filtering Event filtering is the process of correlating incoming events and registered profiles with each other. If an event fits with a profile, i.e. its query evaluates to true, we say the event matches the profile.

Event history We call the internal representation of all events in partial order processed by the system event history.

Consumption mode The consumption mode is a concept concerning the strategy of evaluation in respect to the event history. When specifying a profile, it is necessary to define whether event instances should be disposed of after matching or whether they should be used again for new filtering processes. If disposed of, there are two different possibilities to do so. One way is to delete all event instances which occurred before the matched event instance. This we call *delete*. The other possibility is to only delete those event instances which have really taken part in the matched event instance. This is called *delete & reapply*. If no event instances are deleted, this is called *keep*.

Duplicate handling The term duplicate handling describes which event instances out of a list of identical duplicates are regarded for the filtering process.

We only call those event instances duplicates which have the same event type and are of the same values for their attributes. If we have instances which are interrupted by instances of the same type but with at least one different value, we do not speak of duplicates.

The following possibilities are relevant for our analysis: *first*, *last*, *all*, *n*th, and *n to m*. The values refer to the ordering number of the duplicate events.

Event algebra An event algebra is a meta language which describes the behaviour of events which is specified in profile definition languages [22]. The algebraic structure refers to the combination of several events which thereby form composite events [7]. There are a number of possibilities to define an event algebra. Principally, they all base on the use of operators such as conjunction, disjunction, negation, selection, sequence and simultaneity. Moreover, there is a more detailed description with the help of various parameters and modes. Operators, parameters and modes all describe dependencies between events. For details, we refer to Section 4.1 and 4.2.

3 Survey of Profile Definition Languages and the Systems Using Them

In order to find out whether differing profile definition languages have a different expressiveness or power and follow differing concepts (the problem introduced in Section 1.1), we have analysed filter languages of several systems. Among them, there were not only event notification systems but also event-based infrastructures, active databases, event-action systems and some combined systems.

After a brief description of the intended purpose of a system, we evaluate the parameters for building up composite events such as sequence or conjunction in case the system implements them. Furthermore, we investigate time management, consumption mode and duplicate handling if they were given in the literature about the respective systems.

3.1 Event-based Infrastructures

In the category of event-based infrastructures, we analysed Cobea [34] (which is used e.g. for the management of networks), made an analysis of requirements of the facility management company Lichtvision [20], analysed Rebeca [37] [13] (an event-based architecture for electronic commerce), Regis [36] (a development environment for distributed systems) and Salamander [35] (a system for the distribution of web-applications). An overview of our evaluation is given in Table 2.

Table 2. Event-based infrastructures

System	**Operators**	**Time frame**	**Consumption mode**	**Duplicate handling**
Cobea	Conjunction: *(C1 & C2)* Disjunction: *(C1 | C2)* Sequence: *(C1 ; C2)* Whenever: *($ C1)* Without: *(C1 – C2)*	Yes*:* *Duration*	Keep events	all
Lichtvision	Conjunction Disjunction Sequence Negation	Yes	-	-
Rebeca	Conjunction Disjunction Sequence Negation	Yes	Delete used events and reapply remaining ones: *Chronicle* (choose oldest instances) *Recent* (choose newest instances)	-
Regis	Conjunction**:** *(e1 & e2)* Disjunction: *(e1 | e2)* Sequence: *(e1 ; e2*	Yes: *Dw* (Duration window)	Delete all events	first

	Negation: *({e1; e2}!e3)* Time: *(e + timeperiod)*			
Salamander	only primitive events	-	-	-

Cobea Cobea [34] was developed at the University of Cambridge. It is based on the Corba Event Service, which takes as a foundation the Cambridge Event Architecture (CEA). The system is working according to the *Publish-Register-Notify-Model*, i.e. publishers deliver events whereas users create profiles and are notified whenever these profiles are fulfilled. Cobea is able to filter composite events. Application areas for this system are to be found in the area of distributed systems such as the network management for telecommunication networks.

Operators The event algebra which Cobea uses has been developed in Cambridge. It is structured as follows: The operator without $(C_1\text{-}C_2)$ denotes that C_1 occurs without C_2 having occurred previously. The sequence $(C_1;C_2)$ indicates that C_1 happens before C_2. The conjunction $(C_1\&C_2)$ requires the occurrence of C_1 as well as C_2 in an arbitrary order. $(C_1|C_2)$ represents the disjunction, which needs either C_1 or C_2 to take place. Additionally, there is an operator called whenever $(\$C_1)$. It specifies that each occurrence of C_1 causes a reaction.

Time handling The filter parameters include the *duration*. It describes the timeframe of an event. The other parameters are of no relevance to this work.

Consumption mode and duplicate handling From the definition of sequence and disjunction given in the PhD thesis of Richard Hayton [19] explaining the CEA, we can deduce how the consumption mode and duplicates are handled: The consumption mode is used in the sense of *keep* and duplicates are dealt with in the sense of *all*.

Lichtvision We could learn about an application scenario for composite events in facility management by our cooperation with Carsten Heinrich of the company Lichtvision [20]. This company placed in Berlin, Germany, offers services in the area of electric lighting and facility management.

To date, facility management is working with logic devices such as Lon microcontrollers. They are addressed via a Lon Works network. This is a proprietary system, which works with several programming interfaces. All of them offer different functionality. However, this does not offer a uniform basis for planning.

More often, we can find devices using the bus system of EIB. They are mainly usable for programming them with „and“ or „or“ and are addressed in a hierarchical structure via lines and groups. The number of devices which can be combined is limited, though. Furthermore, these devices do not possess the expressiveness required. Therefore, they are dependent on the inbuilt intelligence of the apparatuses they are connected to. Before their first use, they have to be programmed once.

Such a decentralised use of logic devices causes problems if used for big buildings with more than 30 storeys such as the Treptowers in Berlin-Treptow-Köpenick. Such office buildings have far too many things to control (e.g. blinds, wind sensors, light push-buttons etc.) to still be able to control via the established way. Even in smaller buildings, we encounter problems when their equipment and electrical fittings have to be changed. This only works by totally gutting the building and reinstalling everything in the way it is desired. Moreover, it is difficult to track mistakes as it is not possible to get an overview from a centralised position. Another problem exists regarding the overall planning: light-, clime- and media-technologies are realised by different companies. This is why end-devices, which belong together, are often not connected with each other. In addition, the fitting of electrical cables is complicated as fitters rarely bring together cables according to complicated plans.

These problems could be solved by the deployment of a central server running an event notification service. It would be possible to bring together all cables through one centralised duct to that server. Out of security reasons, a second redundant system would be installed additionally. The desired logic could be realised via the definition of profiles.

An approach such as this would allow for the fast change of the logic installed in a building. With a more powerful language of an event notification service, there would be more possibilities for the design of the fittings and equipment of a building. Cheaper apparatuses could be used since there would be no need for special devices with inbuilt functionalities. Using this approach for the future, it would be possible to have personalised solutions in a building. For example, every user could define the brightness of their room (by the controls for blinds and lamps).

The following operators would be relevant when using an event notification service for facility management:

Operators As can be deduced from the following application scenarios, the conjunction, disjunction and sequence would be relevant.

1. Scenario (time problem): If the use of light push-buttons will not show immediate effect, users will push them again and again. This is why, for a given time interval *t*, after a first push further pushes have to be ignored:

$$P : (e_x \mid (e_{x,} e_x)_t)_t$$

1. Scenario (synchronisation problem): In a room there are two lamps L_1 and L_2. Each has a sensor, S_1 and S_2, measuring the brightness of the room. From a certain threshold, the sensor switches on or off the lamp connected to it. Furthermore, there is a main switch for both lamps together. If only one of the sensors has sent its lamp into another state than that the other lamp is in (i.e. one lamp is switched on and the other off), it is impossible to switch off both lamps with the main switch assuming the control has been programmed in the wrong way. When using a centrally programmable event notification service, this kind of problem can be easily solved:

$$P_1 : ((S_1(brightness < 1000Lux)); T)_t : change(L_1)$$
$$P_2 : ((S_2(brightness < 1000Lux)); T)_t : change(L_2)$$

If a sensor *S* measures no daylight the lamp *L* will be switched off automatically. This is why, only in this case the use of the push-button *T* requires a change of the state to "off".

1. Scenario (Totmann safety lever): All 30 seconds, a wind sensor W_1 sends a signal indicating the wind force. If necessary, the jalousie J_1 can be lifted up (if the wind force is above 9). If the signal does not occur, it may be the case that the wind has destroyed the wind sensor within the last 29 sec because it has become that strong. Then, the jalousie has to be lifted up to be safe:

$$P_1 : (W_1(windForce > 9) \mid \overline{W}_1)_{30\sec} : liftUp(J_1)$$

1. Scenario (central control): When pushing the push-button T_1, the jalousie J_1 has to be lifted up. Furthermore, the janitor is able to centrally lock the building with the control Z_1, i.e. all jalousies of the building are pulled down.

$$P_1 : (T_1(pullUp), Z_1(locked))_\infty : noReaction$$
$$P_2 : (T_1(pullUp), \overline{Z_1(locked)})_\infty : pullUp(J_1)$$

Time handling From the above examples, it becomes clear that we require a timeframe. This also yields for a timestamp.

Rebeca Rebeca [37] [13] is an event-based architecture used in e-commerce that has been developed at the University of Darmstadt. It offers a configurable environment for distributed applications similar to the architecture of the Corba Notification Service. The actual realisation of the system is rudimentary described since publications do not differentiate much between theory and the actual application.

Operators Keeping in mind the above comment, we were able to deduce that the implementation realises the conjunction, disjunction, sequence and negation for composite events.

Time handling Primitive events comprise a timestamp, whereas a time interval is assigned to composite events.

Regis Regis is a development environment for distributed systems realised by the Distributed Software Engineering Group of the University of London. The language GEM [36] extends the inter-process communication enabled in Regis through the integration of composite events.

Operators The conjunction $e_1\&e_2$ is fulfilled if e_1 and e_2 occur, independently of their order. The disjunction $e_1|e_2$ can be matched whenever one of the events occurs. $e_1;e_2$ denotes the sequence and is matched when first e_1 occurs followed by the occurrence of e_2. The negation $\{e_1;e_2\}!e_3$ signifies that e_1 has to be followed by e_2 without e_3 occurring in between the two of them. Moreover, GEM offers a time operator $e+timeperiod$, which informs about the occurrence of e after a certain time span.

Time handling There is the possibility to specify a timeframe in form of a detection window (*dw*). If this detection window is not given, a default value of 12 minutes is used to detect composite events.

There is a differentiation between primitive and composite events: the former carry a timestamp, whereas the latter are assigned a time interval.

Consumption mode and duplicate handling From the examples presented in [36], it becomes evident that for the consumption mode the value “delete” is used. In doing so, they always choose the first duplicate.

Salamander Salamander is a system for the distribution of several internet applications. It was developed by the University of Michigan. In [35], two application are described which deploy Salamander: The IPMA project (Internet Performance Measurement and Analysis) as well as the UARC project (Upper Atmospheric Research Collaboratory). The UARC project is an architecture which enables scientists to jointly research projects even if they are located apart. Furthermore, it makes it possible for scientists to undertake experiments at remote locations such as Alaska or Greenland without actually being there. This is made possible by joint displays of measurements devices and computers which are synchronised. Furthermore, it offers a database for joint comments as well as a distributed text editor. Atmospheric data is transmitted in real time to the scientists taking part in the project. However, Salamander only provides primitive events.

3.2 Active Databases

The active databases we considered were Samos [15] and Sentinel [9] with its filter language Snoop [10]. An overview can be found in Table 3.

Table 3. Active databases

System	Operators	Time frame	Consumption mode	Duplicate handling
Samos	Conjunction: *(E1, E2)* Disjunction: *(E1 | E2)* Sequence: *(E2; E2)* Negation: *(NOT E)*	Timeframe *I* defined by *start_time* and *end_time*: *(E in I)*	Delete used events and reapply remaining ones: *chronicle*	first: *Closure/** nth: *Times (n, E)*
Sentinel	Disjunction: *(E1* $\vee$ *E2)* Sequence: *(E1; E2)* Any: *((I, E1, E2, ..., En), I <= n)* All: *(Any(m, E1, …, En), m<= n)* Periodic: *(P, P*)*		Differing modes of delete and delete & reapply: *recent chronicle, continuous, cumulative*	all last

Samos Samos (Swiss Active Mechanism-Based Object-Oriented Database System) was developed by the Universität Zürich as prototype. Dittrich and Gatziu have described it in [15]. Their goal for the realisation of Samos was to combine features of active databases and of object orientation. The prototype of the system works with the commercial object oriented database system Object-Store. For the detection of events they deploy Petri nets. In its ECA-rules, Samos works with composite events, which are supported by the following operators:

Operators Dittrich and Gatziu offer the disjunction *(E1|E2)*, the conjunction *(E1,E2)*, the sequence *(E1;E2)* and the negation *(Not E)*.

Time handling The occurrence time of an event *E* is denoted by *occ_point(E)*. The operators mentioned above are used in combination with a timeframe *I* in the form *(E in I)*. Normally, it is signified by a start and an ending (*start_time* and *end_time*, respectively). Additionally, there are operators for *overlap* and *extend* to express the intersection and union of two intervals. The start and ending of a timeframe can be explicitly defined or be given implicitly. Then, it would be determined in dependence of other events. If no timeframe is given, it is assumed to be indefinitely. However, this possibility does not exist for the negation.

Consumption mode Samos uses the consumption mode “chronicle”. This corresponds to “delete & reapply”.

Duplicate handling The handling of duplicates can be determined by two operators combined with timeframes.

The c*losure/*-constructor* activates an action once even if the respective event occurs several times within the given time interval. The action takes place after the first occurrence of event *E*. This means that the first duplicate is chosen.

The history-event *TIMES(n,E)* activates an action if event *E* has occurred exactly *n* times within the specified timeframe. This means that the n^{th} duplicate is chosen.

Sentinel Sentinel [9] is an object oriented active database system that has been developed at the Database Systems Research and Development Center of the University of Florida. The profile definition language used for this system is Snoop [10]. It is able to express composite events with the help of the following operators:

Operators The disjunction is expressed by *(E1 $\vee$ E2)*. Within the limits of the publication, they assume the use of an exclusive or. This is done so by presuming that no simultaneous events may take place. Without this presumption the operator is used for an inclusive or. Furthermore, the authors emphasise that it is useful to supply this kind of disjunction with a timeframe. The realisation of this, nevertheless, is not described within their publication.

The conjunction is represented by the operators *Any* and *All*. *Any(I,E1,E2,…,En)* with $I \leq n$ matches exactly if *I* arbitrary events occur out of *n* events independently of their order. *All* is an abbreviation for *Any(n,E1,…,En)* or *Any(m,E1,…,En)* with $m>n$, respectively. If an event occurs repeatedly, this is ignored.

The sequence *(E1;E2)* can be combined with so-called *definite events*. These are events such as the end of a transaction or an absolute time event.

Furthermore, there is a periodic operator. It provides two versions: *(P,P*)*. In *P(E1,[t],E3)*, *E1* and *E3* specify a time span, within which a notification is caused periodically in intervals *t*. In *P**, *t* can be supplied with an additional parameter, which allows a cumulative querying of values within intervals *t*.

Consumption mode Snoop offers four possibilities for the consumption mode: *recent*, *chronicle*, *continuous* and *cumulative*. These differ from the possibilities discussed in Section 2:

Recent means that always the most recent, i.e. the newest, event instance is chosen. Event instances are deleted after a successful matching. *Chronicle* works similarly but chooses the oldest event instances. These two possibilities can be expressed by “delete & reapply” (“recent” in combination with the duplicate handling “first” and “chronicle” with the duplicate handling of “last”).

The possibility *cumulative* chooses all event instances and deletes them and their terminating event instance after a successful filtering. This is the same as the consumption mode “delete”.

The *continuous* consumption mode cannot be expressed by the concepts mentioned so far. It chooses the oldest event instance, and after the successful filtering it only deletes the initiating event instance.

Duplicate handling Additionally to the standard operators mentioned above, there is an aperiodic operator in two versions: *(A,A*)*. For *A(E1,E2,E3)* with *E1*, *E2* and *E3* as arbitrary events, event *A* will be caused for each occurrence of *E2*. The timeframe within which this is filtered, is determined by events *E1* and *E3*. This would be a duplicate handling of “all”.

The version *A*(E1,E2,E3)* differs by the number of activations of *A* as well as by the point in time at which event *A* is activated. *A** cumulates all duplicates of *E2* and initiates *A* when *E3* occurs. This is similar to the duplicate handling of “last”. Nevertheless, the activation of event *A* or the notification, respectively, can have a temporal offset.

Time handling Timeframes can be simulated by temporal events as well as by the *A**-operator.

3.3 Event-Action Systems

We have looked at some event-action systems. They are Active House [3] (a demonstration for the ability of the Cambridge Event Architecture to coordinate several systems in a distributed environment), PLAN [43] (a framework and specification language with an event-condition-action mechanism for clinical test request protocols) and YEAST [30] (a client-server system which enables other systems to register their event-action rules at a central server). A description of the operators and modes is given in Table 4.

Table 4. Event-action systems

System/Function	Operators	Time frame	Consumption mode	Duplicate handling
Active House	Conjunction: *(A & B)* Sequence: *(A; B)* Disjunction: *(A | B),* Negation: *(A - B)* *First(A)*	-	-	first
PLAN	Conjunction Disjunction Negation	yes		first
YEAST	Sequence: *then* Disjunction: *or* Conjunction: *and*	yes: *in,* *within*		first

Active House The active house project [3] has been undertaken by the Opera Group in order to show the possibility to collaboratively work in a distributed environment with several systems using the Cambridge Event Architecture (CEA). In the project, they simulate activities in a house such as activities in the kitchen, ringing of the bell or functionalities of a hi-fi system. Furthermore, an integrated security system monitors the location of people in the house. The activities simulated are sent as events. However, they can also receive events. They are combined to composite events using operators and follow the event-condition-action-concept. This way, it can be specified that whenever the bell is ringing, the toaster has to automatically start roasting a toast or the light has to be automatically switched off.

Operators The Cambridge-Event-Algebra provides the following operators for building composite events: The sequence *A;B* denotes that event *A* has to occur followed by event *B*. The negation *A-B* represents that *A* has to occur without the previous occurrence of *B*. The disjunction operator *A|B* describes that event *A* or event *B* has to occur. The conjunction *A&B* denotes that event *A* as well as event *B* has to occur. The temporal order is insignificant for this. Furthermore, there is an operator called *First(A)*, which tests the first occurrence of event *A*.

Time handling The events are supplied with a timestamp. The problems occurring when dealing with a distributed environment (e.g. determination of a global time) is not solved, however.

Consumption mode The consumption mode is mentioned by the authors, though only as an issue which should be considered when developing an event notification service. Nevertheless, a concrete realisation is not described.

Duplicate handling The system deletes duplicates of an event and notifies once only at the first occurrence of an event.

PLAN PLAN [43] is a framework for controlling the application of given therapy guidelines. It was developed by Bing Wu and Kudakwashe Dube at the Dublin Institute of Technology. The therapy guidelines contain knowledge about diseases, potential therapy possibilities, examinations required for a patient as well as information about a patient's medical history. In a clinical environment, mostly events such as the following are defined: the admission of a new patient, an elapsed time span, receiving new test results of certain medical examinations or a combination of these events.

Potential conditions are the divergence of test results from standard values or the incidence of a specified disease. A potential reaction could be the initiation of certain examinations or tests, the information of a doctor or the notification of a nurse.

The system possesses a specification language, i.e. it can be modified according to the needs of users. It follows the ECA-concept and can be integrated into other technological infrastructures. The events of the ECA-rules can be combined to composite events with the help of the following operators:

Operators Conjunction, disjunction and negation are available.

Time handling Timeframes can be specified in PLAN. This happens using ECA-rules for which the condition has to be set to true. Their actions are initiated by time events.

Duplicate handling Deducing from the examples given in the publication, we consider an implicit use of "first" for duplicate handling.

YEAST Krishnamurthy and Rosenblum describe YEAST in [30]. It is a client-server-system that enables distributed clients to register specifications of event-action-rules for their applications at a central server. This server undertakes filtering and management of incoming events. The event-action-rules can cover several clients. YEAST can process several kinds of composite events. They are defined using the following operators:

Operators The sequence is represented by *then* and the disjunction by *or*. The conjunction is expressed with *and*. However, the conjunction is only filtered successfully if the event instances that are taking part arrive simultaneously.

Time handling Two operators similar to a timeframe are offered: *in* and *within*. They allow the filtering after or within a specified time interval, respectively.

Duplicate handling The parameter `repeat` causes a new registering of a profile whenever it has been matched once. It is implied that the first event of a sequence of duplicates is considered, except the `repeat` parameter is used. Then, all duplicates which match a profile are considered.

3.4 Event Notification Services

The event notification systems we have investigated are CompAS [29] (prototype of an event notification system for composite events which was developed within the project MediAS at Freie Universität Berlin), the Corba Notification Service [1], Elvin [40] [14], Hermes [12] (an event notification system for digital libraries), Keryx [6] (which is designed to distribute notifications in the internet), READY [17] [18] (which is the sequel of the event-action system YEAST) and Siena [8]. Table 5 gives an overview of the operators and modes of the event notification systems.

Table 5. Event notification systems

System	Operators	Time frame	Consumption mode	Duplicate handling
CompAS	Conjunction	yes	keep	first
	Disjunction		delete	last
	Sequence		delete & reapply	all
	Negation			nth
	Selection			from n to m
Corba	Only primitive events	-	-	-

Notification Service				
Elvin	Only primitive events	-	-	-
Hermes	Only primitive events	-	-	-
Keryx	Only primitive events	-	-	-
READY	Conjunction: *(Ev_x && Ev_y)* Disjunction: *(Ev_x || Ev_y)* Sequence: *(Ev_x ; Ev_y)* Negation: *(not Ev_x)*			first last all n^{th} from n to m
Siena	Sequence: $A \cdot B$	-	delete	first

CompAS CompAS [29] is a system that has been developed at the Institut für Informatik at Freie Universität Berlin. It is a system which is related to the work described in this technical report. It is an event notification service, which deals with composite events. The version DAS [5] realises this for a distributed environment.

Operators The following event operators are implemented in the system: conjunction, disjunction, sequence, negation, selection and simultaneity.

Time handling Event instances are supplied with a timestamp. Composite events have a timeframe within which they have to occur.

Consumption mode For the consumption mode the possibilities “delete”, “delete & reapply” as well as “keep” are realised.

Duplicate handling It is possible to select the first, last, i^{th} and all duplicates of an event.

Corba Notification Service The Corba Notification Service is an event notification service which is based on Corba. Corba (Common Object Request Broker Architecture) is a specification of the Object Management Group. It offers a possibility for communication among arbitrary applications in a distributed environment. The event notification service separates the communication between Corba-objects and offers an event-based asynchronous exchange via so-called notification channels. These are responsible for the filtering of the events. Primitive events are the only ones that are supported [1].

Elvin The works in [40] and [14] describe the Elvin system, which has been developed by the Distributed Systems Technology Centre. It is an event notification service that uses a distributed client-/server-architecture. Elvin itself acts as a server, whereas the publishers and users work as clients. Neither Elvin nor the next development Elvin4 support composite events.

Hermes Hermes is an event notification service that has been developed at the Institut für Informatik at Freie Universität Berlin and is used for digital libraries [12]. The service offers a uniform interface for users which enables them to be notified about literature of differing publishers. Even publishers who do not offer an own notification service are supported. Users have the option to sort the documents presented in a notification by a relevance feedback. Hermes supports primitive events only.

Keryx The event notification service Keryx has been developed by Keryxsoft, which is a research group of the Hewlett Packard Laboratories in Bristol. It is a successor of the Nexus Event Service. The service is realised independent of languages and platform and uses a client-server structure. Its functionality is to distribute notifications throughout the Internet. From [6] it can be concluded that there is no possibility for Keryx to build composite events, i.e. they only offer primitive events.

READY READY, which is described in [17] and [18], is a successor of the YEAST event-action-system. It has been developed by AT&T and is an asynchronous event notification service. READY offers a middleware for applications which have been developed separately. Publishers and users can be integrated independently from each other into the system. The following operators are used for the realisation of composite events:

Operators The conjunction is expressed by *Ev_x&&Ev_y*, the disjunction using *Ev_x||Ev_y*, the sequence by *Ev_x;Ev_y* and the negation via *notEv_x*.

Duplicate handling READY uses an expression for matching intervals of duplicate occurrences. This expression offers the possibility to specify how often an event has to match in order for a notification to be sent. It is possible to state the following expressions: *"at most x"*, *"exactly x"*, *"x to y"* and *"at least x"*.

Examples for this are *Ev1[0..3]* with at most 3 matchings, *Ev1[3..5]* with 3 to 5 matchings, *Ev1[3..3]* with exactly 3 matchings and *Ev1[3..*]* with at least 3 matchings. Here "0" and "*" denote there is no upper or lower bound. Furthermore, it is possible to specify the first, last, i*th* and all duplicates.

Siena Siena [8] is the prototype of an event notification service developed by Carzaniga at the University of Colorado. The system is the product of a research project targeting the development and the implementation of a scalable event notification service using a distributed architecture of event filtering brokers. This means that Siena can be used internet-wide.

In the scope of this research project, there have been developed a number of fundamental theoretical approaches for event notification services in distributed environments. Nevertheless, the system is realised only rudimentary. It can filter composite events.

Operators Siena only implements the sequence $A \cdot B$. It selects the first A_i^j followed by the first B_k^m, if $i<k$ and $j<m$ holds.

Time handling Each event $E_{t_i}^{t_j}$ contains two timestamps. The former t_i it receives at the creation of the event, the latter t_j at the time of filtering.

Consumption mode and duplicate handling Carzaniga gives the following example for clarifying his filtering process: Given the following event history:

$$B_4^1\ A_3^2\ A_1^3\ B_2^4\ A_5^5\ B_6^6\ A_7^7\ B_8^8$$

We want to express the sequence $A \cdot B$: The first sequence that is selected is $A_3^2 \cdot B_6^6$. The next sequence is $A_7^7 \cdot B_8^8$. This implies a consumption mode of "delete" and a duplicate handling of "first" (first duplicate).

3.5 Combined Systems

We examined further systems: Conquer [33] and OpenCQ [32] [39] (systems which are able to handle web-documents, databases and files), and also Eve [16] [41] (which combines characteristics of active databases and event-based architectures in its task to execute event driven workflows). An overview is given in Table 6.

Table 6. Combined systems

System	Operators	Time frame	Consumption mode	Duplicate handling
CQ: Conquer and OpenCQ	Sequence: *(E1; E2)* Simultaneity: *(E1|| E2)* Conjunction Disjunction Negation	Yes	-	-
Eve	Conjunction: *(CON(ET1, ET2, sw))* Disjunction: *(DEX(ET1, ET2))* Sequence: *(SEQ(ET1, ET2, sw))* Simultanity: *(CCR(ET1, ET2, sw))* Negation: *(NEG(ET1, (ET2, ET3, sw), sw))* Repetition: *(ET1, times, sw)*	Yes	delete & reapply: *chronicle*	First nth

CQ: Conquer and OpenCQ Conquer and OpenCQ have been developed at the Oregon Graduate Institute of Science and Technology. Conquer is discussed in [33] and OpenCQ in [32] by Liu, Pu and Tang. Further information is available from [39]. Conquer is a system for monitoring changes in web documents. OpenCQ extends the possibilities of Conquer: It can not only monitor web documents but also databases and files.

Both systems are based on the CQ-specification language. This can express composite events. The operators presented in their publications vary. In the following, we will present all of them:

Operators The sequence is expressed by *E1;E2*. It is fulfilled whenever *E1* occurs before *E2*. The simultaneity *E1||E2* matches when *E1* and *E2* occur at the same time. Furthermore, there are the conjunction, disjunction as well as the negation.

Time handling It is possible to specify a timeframe by choosing a time interval within which a profile is valid. This interval can be defined by a start and ending. Another possibility is to only name the ending when registering the profile. Then, for the beginning the time of the registration of the profile is chosen automatically.

Eve Eve is an event engine that was developed at Universität Zürich. It realises an event-driven execution of distributed workflows ([16] and [41]). This means that the engine is able to register events, recognize and administer events as well as notify distributed autonomous software components, which represent workflows. The system combines ECA-rules with active database mechanisms in an integrated event-based architecture. In order to achieve this, it is able to form composite events with the following operators:

Operators ET1, ET2 and *ET3* are event types. The conjunction *CON(ET1,ET2,sw)* is recognised whenever *ET1* and *ET2* occur regardless of their relative order. *sw* denotes the restriction that both events have to occur within the same workflow. *DEX(ET1,ET2)* expresses the exclusive disjunction, which is matched if only one of the events has occurred. The sequence *SEQ(ET1,ET2,sw)* is matched when *ET1* temporally occurs before *ET2*. The simultaneity is represented by *CCR(ET1,ET2,sw)*. It is fulfilled whenever both events take place at the same time.

The authors are dealing with time according to the 2g-precedence-model. This model defines within which time limits we can still accept the occurrence as simultaneous. *NEG(ET1,(ET2,ET3,sw),sw)* represents the negation. It tests whether *ET1* does not occur within the interval given by *ET2* and *ET3*. Furthermore, they offer an operator representing the repetition *REP(ET1,times,sw)*. It tests whether *ET1* occurs as often as defined by the parameter *times*.

Time handling The definition of an event can be restricted by the definition of a timeframe, which is given by two events. Events have a large number of attributes that specify them more closely. Among them are a timestamp and an identifier that assigns the membership of the event to a workflow. The timestamp of the negation is given by the timestamp of the event that terminates the negation.

Consumption mode In order to work properly, Eve requires a consumption mode of “chronicle”. This is equivalent to “delete & reapply”.

Duplicate handling The publication implies that the first duplicate of an event is selected. Furthermore, the repetition operator allows specifying a kind of filtering that searches for a specified number of events that cumulatively occur. This is the same as a notification after the i^{th} duplicate.

4 Classification of Filter Languages - a Comparative Study

On grounds of our previous analysis, we have classified selected applications and event notification systems by ordering them according to the power of their profile definition languages. For doing so, we have developed classification criteria which base on two works [44] [26] for the description of the semantics of filter languages.

Looking at the analysed systems, it becomes clear that to simply consider the operators is not sufficient in order to convey the full semantics. Each system offers parameters, which more closely define the operators, or operators, which carry differing semantics to others. Nevertheless the exact description of these operators/parameters is rarely given in literature. Despite that, we can summarize the operators to: conjunction, disjunction, negation, selection, sequence und simultaneity.

Unland and Zimmer give in [44] a very detailed description of parameter-semantics and their modes. For example, they have defined a concurrency and a consumption mode which are presented in Table 7.

Table 7. Concurrency and consumption mode (Unland/Zimmer)

Parameter	**Mode**	**Function**
Concurrency	Overlapping	Components of different event instances may overlap each other
	Non-overlapping	Components of different event instances may not overlap each other
Consumption	Shared	No event instance is deleted
	Exclusive parameter	All event instances which have taken part in the matching of a composite event are deleted
	Exclusive	All event instances before the terminating event instance of a composite event are deleted

Some of the modes they present are dependent on each other, so not all combinations are possible. If we have a consumption mode of shared, the concurrency mode has to be overlapping; a consumption mode of exclusive only is logical in combination with a concurrency of non-overlapping. Furthermore, some modes of parameters they give do not occur in the kind of systems we are interested in (e.g. concurrency mode). Also the traversion mode, which describes the direction of traversing composite events, is irrelevant for us since systems filter their events in timely order and not backwards. So, we neglect this parameter.

Due to this and the fact that we do not have the exact parameters of all systems at hand, we choose the description of Hinze and Voisard [26] as a basis for the classification of our systems. Please refer to Table 8.

Table 8. Comparison classifications of Hinze/Voisard and Unland/Zimmer

Hinze/Voisard	**Unland/Zimmer**
Composite events	
Operators	
Timeframes	
Consumption Mode	
Duplicate handling	Coupling Mode
	Concurrency Mode
	Parameter Selection
	Traversion Mode

The only change is that the consumption mode we chose is that used by Unland/Zimmer. They offer three values as can be seen in Table 6. Unland/Zimmer's coupling mode can be neglected since it may be expressed with the help of the negation operator and wildcards. Unland/Zimmer's parameter selection is expressed via Hinze/Voisard's duplicate handling.

4.1 Composite Events (Unland/Zimmer)

Notions Unland and Zimmer have developed their concepts for the description of composite events within the scope of active databases [44]. Due to this, their terminology mainly is limited to that particular area as well. This is why, they are using some of the notions differently than we do in this technical report so far. In the following please find a summary of their definitions:

Event An event is simply seen as the occurrence of a situation which requires an action.

Event-Condition-Action-Rule At the occurrence of an event which matches a specification of an event type, the condition of the rule is tested. If applicable an action which previously had been defined is executed.

Time domain For their project, Unland and Zimmer have assumed an equidistant discrete time domain.

Event instance An event instance represents a concrete event existing in the real world.

Event type (E_i) The event type is an abstract description of a set of event instances which can be described by more closely defined similarities. This can be the same describing parameters or the same reaction specified for the occurrence of an event. As an example, we could have the following description:

$$E_4{:=};(\textit{last: shared}(E_1),\ \textit{first: exclusive parameter}(E_2),\ \textit{shared}(E_3))$$

The event type E_4 is specified by the sequence of E_1, E_2 and E_3. They are filtered with a different kind of consumption mode: E_1 and E_3 are repeatedly used in the filtering process whereas E_2 is deleted after matching once.

Parameter Parameters are used for the specific description of event instances. They give details about their event type, the point in time when they occur as well as about their activator.

Primitive event types Unland and Zimmer differentiate primitive event types between *external*, *temporal* and *database-specific event types*. The latter refer to data manipulation or transaction operations. The temporal event types split up into *absolute*, *relative* and *periodic time event types*. External event types represent events which occur external to the database or the computer.

Complex event types Unland and Zimmer call the combination of event types through operators of an event algebra a complex event type. The components of a complex event type are called e*vent type components* E_{ij} – they can be of primitive or complex type. The complete formation is referred to as *parent event type*.

Event instance sequence (EIS^{E_i}) An event instance sequence is a partial ordered set of event instances, which have occurred in the system. They are ordered in respect to their time of occurrence. One can differentiate between *instance-oriented event instances* and *type-oriented event instance sequences*. Instance-oriented event instance sequences represent these kinds of event components, which have caused the occurrence of complex events. Whereas type-oriented event instance sequences describe the event components required for the recognition of complex events. An example for their notation is the following event instance sequence:

$$EIS^1 := ei_1^1 \ ei_1^2 \ ei_1^3 \ ei_2^1 \ ei_1^4 \ ei_2^2 \ ei_3^1$$

The event instance sequence EIS^1 is composed of three instances of type E_1, followed by one of type E_2, by one of E_1 and one of E_2, finalized by an instance of type E_3.

Event instance pattern Event instance patterns describe the same constituents as type-oriented event instance sequences, i.e. the event components required for the recognition of a complex event type.

Initiator The event instance of an event instance sequence which starts the recognition of a complex is called the *initiator*. The event instance which ends the event instance sequence is the *terminator*. Those which lie in between are referred to as interiors.

Event group An event group is formed by those event instances which are ended by the exact same terminator.

Instance set A duplicate set is formed by different event instances of the same event type.

Operators The following operators are used by Unland and Zimmer:

$\wedge$*-operator* The conjunction operator requires the occurrence of the instances of the events concerned in arbitrary order.

$\vee$*-operator* For the disjunction operator, it is sufficient if at least one of the given instances takes place.

$\neg$*-operator* The negation operator describes the absence of an instance specified within a given time interval. This time interval is defined by the occurrence time of the first operand (start time of the interval) and the occurrence time of the last operand (ending time of the interval) of the negation operator.

;-operator The sequence operator requires the occurrence of the specified instances in the order defined.

==-operator The simultaneity operator describes that event instances have to take place at the same point in time.

Operator modes The authors differentiate between three modes for the specification of the operator relationships: they give the traversion mode, the coupling mode and the concurrency mode.

Traversion mode There are two different traversion modes: from left to right and vice versa. The direction is highly relevant for the evaluation. In the following example, given the event instance sequence EIS^2

$$EIS^2 := ei_1^1\ ei_2^1\ ei_1^2\ ei_2^2\ ei_2^3\ ei_1^3\ ei_3^1$$

and the event type E_4

$$E_4 :=\ ;\ (last: E_1, first: E_2, E_3)$$

we have different results depending on the traversion mode (direction of evaluation). Using the traversion from right to left, at first the first instance of type E_3 (ei_3^1) is chosen. Then the first instance of the event type E_2 (ei_2^1) and finally the last instance of the type E_1 (ei_1^1), which has to occur before ei_2^1. So we get: $(ei_1^1,\ ei_2^1,\ ei_3^1)$. Whereas using the traversion mode from left to right, we gain the following sequence: $(ei_1^2,\ ei_2^2,\ ei_3^1)$. At first, the last instance of type E_1 is chosen – this would be ei_1^3. This is no valid result, however. This is due to the fact that between ei_1^3 as initiator and ei_3^1 as terminator there is no instance of the event type E_2. This is why we have to perform backtracking to ei_1^2. Next, the first instance of type E_2 is looked for, which follows ei_1^2: ei_2^2. We terminate with ei_3^1.

Coupling mode The coupling mode determines whether it is allowed that the searched event instances are interrupted by other event instances that are irrelevant for the event instance pattern or whether they may not be interrupted. Given the following event instance sequence

$$EIS^3 := ei_1^1\ ei_2^1\ ei_3^1$$

and event type

$$E_5 :=\ ;\ (E_1, E_3)$$

If we have a *continuous* coupling mode, no event of type E_5 would be matched since ei_2^1 interrupts the sequence ei_1^1 and ei_3^1. Whereas if we have a *non-continuous* coupling mode, this interruption would be allowed and an event of type E_5 would be recognised.

Concurrency mode The concurrency mode determines whether event instance components may overlap in respect to their occurrence time: Here, we have an *overlapping* concurrency mode. An alternative would be a non-overlapping concurrency mode. Consider event instance sequence

$$EIS^4 := ei_1^1\ ei_1^2\ ei_3^1\ ei_2^1\ ei_3^2$$

and event types

$$E_5 := \ ;\ (E_1, E_2) \text{ and } E_6 := \ ;\ (E_3, E_5)$$

Then, the occurrence time of e_5^1 is identical with the occurrence time of the terminating event instance component e_2^1. Thus, instance e_3^1 of type E_3 has an earlier occurrence time than e_5^1, but a later one than the initiating event instance sequence e_1^1 or e_1^2, respectively (depending on the strategy of evaluation). So, whether in an event instance sequence EIS^4 an event e_6^1 will be recognized, is dependent on the concurrency mode of that event, i.e. whether it is defined as "overlapping".

Modes of operands Also the operands offer a range of specification possibilities:

Parameter selection Unland and Zimmer describe two kinds of duplicate handling – one only uses the minimum of the event instances required. Whereas the other chooses more than the number prescribed by one delimiter. *First* and *last* refer to the instance or instances of an event with the oldest and youngest timestamp, respectively. So they are strategies choosing a minimal number. Cumulative selecting strategies are the *cumulative* and *restricted cumulative* parameter selection. The parameter "cumulative" selects all instances of an event type, whereas "restricted cumulative" only chooses those event instance components which are required by the event instance pattern of the parent type. Additionally, the authors define two modes: *combinations* and *combinations minimum*. We need these modes in order to make it possible that different duplicate sets of an event instance component can be combined subsequently with the duplicate sets of other event type components.

Consumption mode The recycling of event instances when matching complex events can be specified through the consumption mode by using "*shared*", "*exclusive parameter*" and "*exclusive*". "Exclusive" deletes all event instances which occur prior to the terminating event. If we look for $E_5 := ;(E_1, E_2)$, then the event instances selected would be those depicted in Fig. 2 [1].

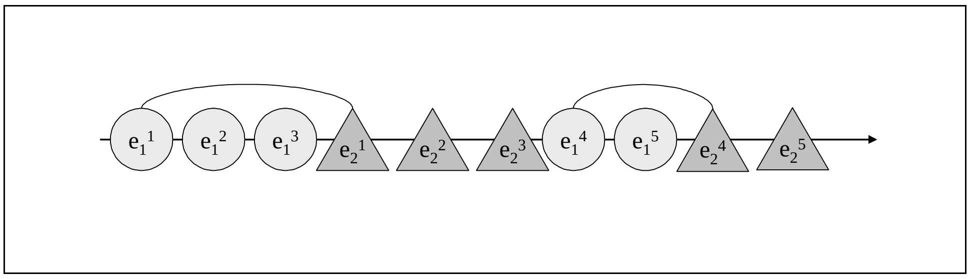

Fig. 2. Exclusive consumption mode

A consumption mode of "exclusive parameter" deletes those event instances which have already been used for the matching of a complex event. Refer to Fig. 3.

[1] In Fig. 2, we present the event history as temporal sequence from left to right.

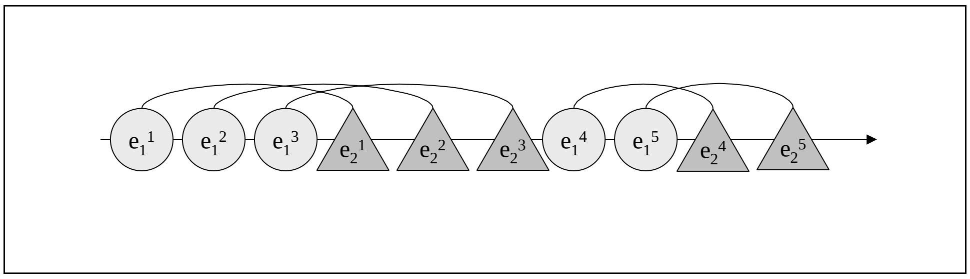

Fig. 3. Exclusive parameter consumption mode

In opposition to that, a consumption mode of “shared” deletes no event instances at all (Fig. 4).

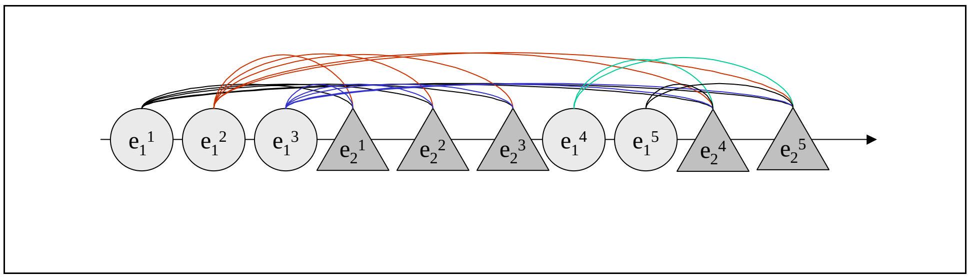

Fig. 4. Shared consumption mode

Delimiter The number of duplicates of an event type that are taken into account for matching is specified by a number or an interval.

The formal specification system of Unland and Zimmer is very comprehensive and allows for the detailed description of the semantics of complex events. Nevertheless, there is the disadvantage that certain issues are subdivided into components which in itself should not be possible to subdivide. This is due to the fact that they are dependent on each other. An example for this problem are the concurrency mode and consumption mode which are interconnected: If we choose the parameter “exclusive” for the consumption mode, then this makes the choice of “overlapping” for the concurrency mode impossible. The reason for this is in the deletion of all event instances from the event history which occur prior to the terminating event. Moreover, the consumption mode “shared” involves an overlapping, as one event instance is used for several matchings. Only the possibility of “exclusive parameter” for the consumption mode does not influence the choice of the concurrency mode. If we have $E_4{:=};(E_1,E_3)$ and $E_5{:=};(E_4,E_2)$ and are looking for E_5, we only receive one match (Fig. 5).

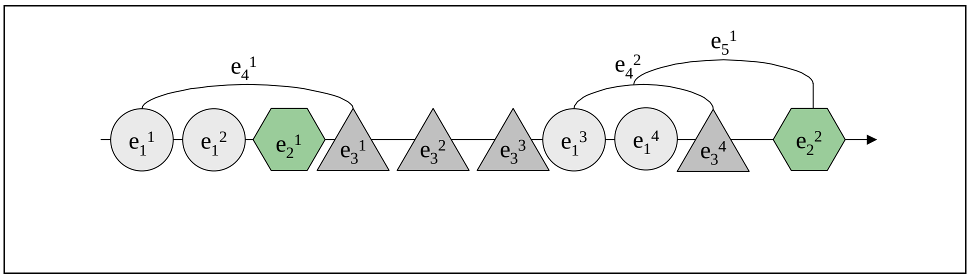

Fig. 5. Consumption exclusive and concurrency overlapping

This match is (e_1^3, e_3^4, e_2^2) or (e_4^2, e_2^2) as there is no matching instance of E_2 for the first occurrence of instance E_4. This is due to the fact that e_2^1 is already deleted because of the “exclusive” consumption mode. The following dependencies result from the possibilities of the parameters for the consumption and concurrency mode (Table 9).

Table 9. Dependency of consumption and concurrency mode

Consumption Mode	Concurrency Mode
shared	overlapping
exclusive parameter	undetermined
exclusive	subsequently

If we have a consumption mode of "exclusive", it is reasonable to use the subsequent concurrency mode since this consumption mode excludes overlapping events. Moreover, the consumption mode "shared" would not be reasonable with a subsequent concurrency mode as overlappings are the typical feature of the "shared" consumption mode.

4.2 Composite Events (Hinze/Voisard)

Notions We here will introduce the notions used by Hinze and Voisard in [26] unless they have already been introduced in Section 2. For the definitions the following holds: (e_i), $i \in N$ denotes primitive as well as composite events. For $t(e_i)$, t specifies the timestamp of an event and for $(e_i)_t$, t represents a timeframe.

Event (e_i) This notion refers to the previous definition of an event to be found in Section 2.

Event class (E_i) The event class of Hinze and Voisard also refers to the definition which can be found in Section 2.

Event instance We extend the previous definition: The event instance (e_i) of event class (E_i) is denoted by $e_i \in E_i$. It is possible that an event instance (e_i) belongs to several event classes, i.e. $e_i \in E_i$ and $e_i \in E_m$ hold at the same time. An event class can have subclasses $e_i \in E_i \subset E_m$.

Primitive event Primitive events are divided into *time events* and *content events*. Time events describe the occurrence of a point in time whereas content events denote all changes of states.

Composite event This concept corresponds to Unland and Zimmer's concept of complex events types.

Timestamp The timestamp $t(e)$ of an event $e \in E_i$ stands for the occurrence time of this event.

Duplicates We follow the definition which has already been introduced earlier in this technical report.

Event operators Hinze and Voisard describe the meaning of their event operators as follows:

Disjunction The disjunction of two events ($e_1|e_2$) signifies the occurrence of at least one of these events. The occurrence point of the composite event $e_3=(e_1|e_2)$ is the occurrence point of the earlier event taking part in the disjunction $t(e_3):=min\{t(e_1),t(e_2)\}$.

Conjunction For the conjunction of two events $(e_1,e_2)_t$, we have to specify a timeframe for the filtering of the profile. Both of the events have to occur within the given time span described by the timeframe. The occurrence point of the composite event $e_3=(e_1,e_2)$ is the occurrence point of the later one of both of the events taking part in the conjunction $t(e_3):=max\{t(e_1),t(e_2)\}$.

Sequence The sequence of two events $(e_1;e_2)_t$ tightens the conditions of the conjunction: first, e_1 has to occur followed by e_2. The occurrence point of the composite event $e_3=(e_1;e_2)$ is the same as that of the second event e_2. So, it holds: $t(e_3):=t(e_2)$.

Negation The negation of an event $(\overline{e})_t$ informs about the fact that this event has not taken place within the given timeframe $[t_{start},t_{end}]$, $t_{end}=t_{start}+t$. The occurrence time of the negative event $(\overline{e})_t$ is the ending of the time interval $t((\overline{e})_t) := t_{end}((\overline{e})_t)$.

Selection The selection $e^{[i]}$, *with* $i \in \mathbb{N}$ denotes the occurrence of the i^{th} event within a list of event instances.

Basic principles of parameters Hinze and Voisard remark that the semantics of temporal event operators is dependent on the applications which are chosen. This is why, they introduce special parameters which allow for this matter. The modes relevant for this are *event selection, event instance pattern*, *event instance selection* and *event consumption*.

The concept of event selection, i.e. the mechanism describing how primitive events are chosen, is not described in detail by the authors but assumed to be attribute-value-pairs. The event instance pattern, i.e. the operator concept underlying the composition of composite events has been discussed in the previous section.

The event instance selection deals with the selection of duplicates of events. It allows for the consideration of application-dependent circumstances of this selection. All possibilities for this selection are depicted in Table 10.

Table 10. Example for event instance selection

Event instance selection	**Application example**
first	Changes of state happen once only, so all further duplicates are redundant.
i^{th}	For technical readings where a high precision is required, it might be interesting not only to consider the first but also the second and third reading, when the parameters have passed a certain threshold. Also, in the medical area it is possible that the first occurrence of an event does not indicate anything unusual; nevertheless if that value occurs repeatedly, this indicates a problem.
last	For location systems we usually are only interested in the last duplicate of an event since only that one can give valid information about the location of an object.
all	Security systems (movement sensors, messages about doors which are open within specified time intervals) have to regard all duplicates since every discrepancy can represent a security hazard.

Hinze and Voisard express the event instance selection via parameters for event operators. This will be further described later this section (event operators with parameters).

The event consumption determines whether event instances which have been considered for the filtering process will be deleted from the event history or whether they will be kept for the filtering of further composite events of the same event class, i.e. they would be recycled. The authors differentiate between two kinds of consumption modes:

All pairs All event instances are kept, i.e. even those which have been used in the filtering process before. That way, we gain all possible permutations of event instance combinations. Unland and Zimmer call this the “shared” consumption mode (see Fig. 4).

Unique pairs All event instances that have been used in the profile matching of a composite event are deleted. However, then the event history is refiltered using the remaining event instances. This is identical to the mode “exclusive parameter” of Unland and Zimmer (see Fig. 2).

Parameter definitions The following definitions of notions are required for the parameters of the event operators:

Profile-matching An event *e* matches a profiles *p* whenever the attributes of the event are the same as all the attributes of the profile: $p \prec e$. This notation of matching operators refers to its use by Carzaniga, Rosenblum and Wolf in [8].

Event space The set of all events that can possibly occur in a system is called event space $\mathbb{E}$. The set of all time events is referred to as $\mathbb{E}_t$.

Trace A trace $tr_{t1,t2}$ is a sequence of temporally ordered events with the beginning *t1* and the ending *t2*. Thus, the event history of a service (containing events $e \in \mathbb{E}$) is the trace $tr_{t0,\infty}$. *t0* is the beginning, i.e. the point in time when the event notification service started filtering events. A trace is like a list *L*. It is possible to access the elements via a list index: *L*[*i*], $i \in \mathbb{N}$ yields the i^{th} element.

Trace view A trace view *tr*({E_1}) is a part of a sequence or a sub list of a trace *tr*, which contains only event instances of one single event class, i.e. $e \in E_1$ is contained in a partial order.

Trace renumbering A trace *tr* can be divided into trace views *tr*[*1*], … , *tr*[*n*]. These contain event instances of the same class. These event instances are numbered in the following way: *tr*[*x*,*y*], $x \in \mathbb{N}$ as number of subsequences and *y*∈[*1*, *length*(*tr*[*x*])] as index number of the event instance within the respective subsequence.

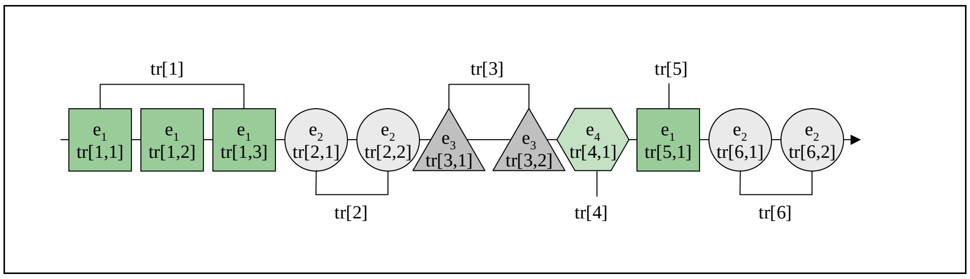

Fig. 6. Example for trace renumbering

Fig. 6 illustrates an example for trace renumbering. There the membership in the same class is denoted by using the same name. The first trace view *tr*[1] is built from three event instances *tr*[1,1], *tr*[1,2] and *tr*[1,3] since all belong to the same event class E_1. Then, an instance of another event class follows: E_2. This is why, the next trace view *tr*[2] begins with *tr*[2,1]. This trace view consists of the event instances *tr*[2,1] and *tr*[2,2]. The other trace views are built analogously. Event instance *tr*[5,1] can also be referred to as trace view *tr*[5] even though it belongs to event class E_1 (same as *tr*[1]). This is due to the fact that a repetition of event instances is independent of instances of the same class prior to that particular instance if these instances are separated by instances of other classes.

Event operators with parameters

Operators For event operators with parameters, we assume without loss of generality as precondition that our traces always start with e$_1$. The operators are defined as follows:

Negative event

$$(\overline{p}_1)_{t1} \prec e_t := \{\forall e_1 \in \mathbb{E} \wedge t(e_1) \in [t(e_t) - t_1,\ t(e_t)]:\ \neg(p_1 \prec e_1\ )\ \}$$

Time event e_t matches the negative profile of p_1 within time span t_1 if, within time span $t(e_1) - t_1$ until $t(e_t)$, none of the occurring events matches profile p_1.

$$(\overline{p}_1)_{t1}\ (tr)\ \prec\ \{e_t \mid e_t \in tr\ \wedge\ (\overline{p}_1)_{t1} \prec e_t\}$$

The negation of profile p_1 within time interval t_1, applied to trace tr, is the set of all time events out of this trace tr, which match the negation of profil p_1 within time interval t_1.

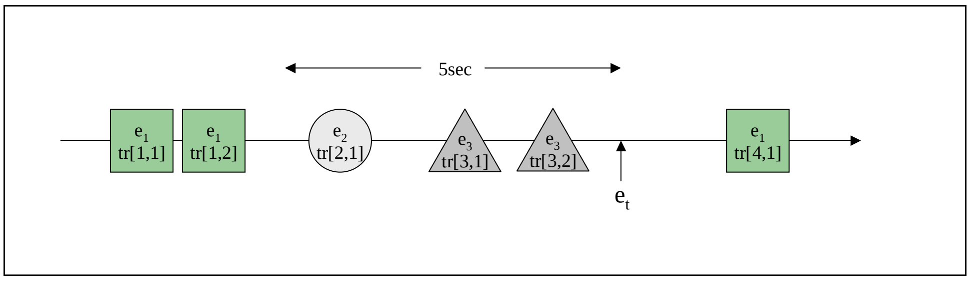

Fig. 7. Negation $\overline{(p_1)}_{t1} := \overline{(e_1)}_{5\,\text{sec}}$

The example depicted in Fig. 7 describes the test whether within time frame t_1=5*sec* profile p_1 is not fulfilled, i.e. whether no event e$_1$ takes place. Since only event instances of classes E$_2$ and E$_3$ occur $\overline{(e_1)}_{5\,\text{sec}}$ is true.

Disjunction For an event $e \in \mathbb{E}$ and profiles p_1 and p_2 holds:

$$(p_1 \mid p_2)\ \prec e\ :=\ \{p_1 \prec e\ \vee\ p_2 \prec e\}$$

$$\begin{aligned}(p_1 \mid p_2)\ (tr) \prec \{tr[x,z] \mid tr[x,z]\ \in\ & tr(e_1, e_2)\\ & \wedge\ (p_1 \mid p_2) \prec tr[x,z]\\ & \wedge x \in [1,\infty),\ z \in [z_{\min}, z_{\max}]\}\end{aligned}$$

We receive the set of events of a trace tr, which match a disjunction of p_1 and p_2, by choosing all elements between z_{min} and z_{max} (which match p_1 or p_2) in the respective trace views $tr[x]$. If the same value is selected for z_{min} and z_{max}, for each trace view only one event instance of the same class will be selected (i.e. that one that has been specified by the value of $z_{min}=z_{max}$). Otherwise, z_{min} and z_{max} select several duplicates of the same event instance which follow subsequently and lie within the range given by z_{min} and z_{max}.

An example for a disjunction $p_3 = (p_1 \mid p_2)$ with $p_1 \prec e_1$ and $p_2 \prec e_2$ as well as $z_{min}=1$ and $z_{max}=1$ is depicted in Fig. 8:

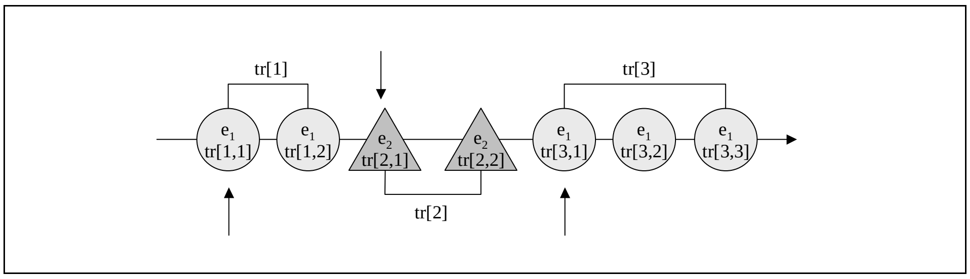

Fig. 8. Disjunction $p_3:=(e_1|e_2)$, $z_{min}=z_{max}=1$

The same example with different z_{min} and z_{max} would choose other, and in the case of $z_{min}=2$ and $z_{max}=4$, more duplicates. The second duplicate of *tr*[1], *tr*[2] and *tr*[3] as well as the third duplicate of *tr*[3] would be selected. If there were longer trace views, the fourth duplicate out of them would be chosen as well (see Fig. 9):

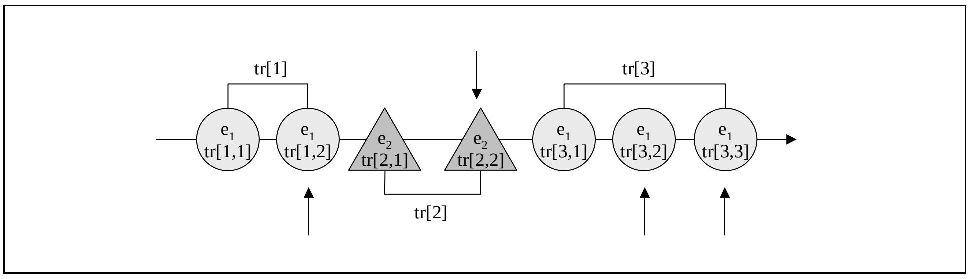

Fig. 9. Disjunction $p_3:=(e_1|e_2)$ with $z_{min}=2$ and $z_{max}=4$

Conjunction For profiles p_1 and p_2 as well as events $e_1, e_2 \in \mathbb{E}$ holds:

$$(p_1, p_2)_t \prec (e_1, e_2) \;:=\; \{p_1 \prec e_1 \;\wedge\; p_2 \prec e_2,\; |t(e_2) - t(e_1)| \le t\}$$

$$\begin{aligned}(p_1, p_2)_t(tr) \;\prec\; \{(tr[x,z],\; tr[y+1,w])\,|\\ &(tr[x,z],\; tr[y+1,w]) \;\in\; tr(e_1,e_2)\\ &\wedge (p_1, p_2)_t \prec (tr[x,z],\; tr[y+1,w])\\ &\wedge\; x \in [1,\infty) \wedge\; y \in [1,\infty)\\ &\wedge w \in [w_{\min}, w_{\max}] \wedge\; z \in [z_{\min}, z_{\max}]\\ &\wedge\; P_{xy}\}\end{aligned}$$

The set of the events of traces *tr* which match the conjunction of p_1 and p_2 consists of the tuples (e_1,e_2), which match (p_1,p_2), and for which the events e_1 or e_2 occur at certain places of the respective trace views. This means, $z \in [z_{min}, z_{max}]$ determines the duplicates which are chosen for $e_1=tr[x,z]$ and $w \in [w_{min}, w_{max}]$ those for $e_2=tr[y+1,w]$. P_{xy} specifies as condition the relation between *x* and *y*. This conveys which pairs are selected for (p_1,p_2). An example for the conjunction is depicted in Fig. 10:

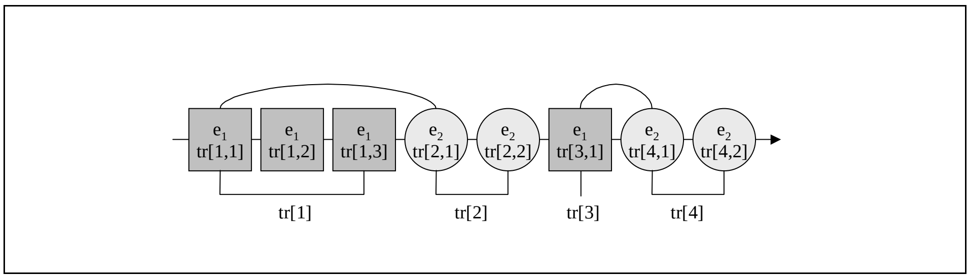

Fig. 10. Conjunction $p_4:=(e_1,e_2)$, $z_{min}=z_{max}= 1$, $w_{min}=w_{max}=1$ as well as $P_{xy}:(x=y)$

The value of P_{xy} determines that only unique pairs have to be selected, i.e. each event instance may be used for the conjunction once at the most. $z_{min}=z_{max}=1$ specifies that for each trace view *tr*[*x*,*z*], we only consider the first duplicate and $w_{min}=w_{max}=1$ works analogously for *tr*[*y*+1,*w*].

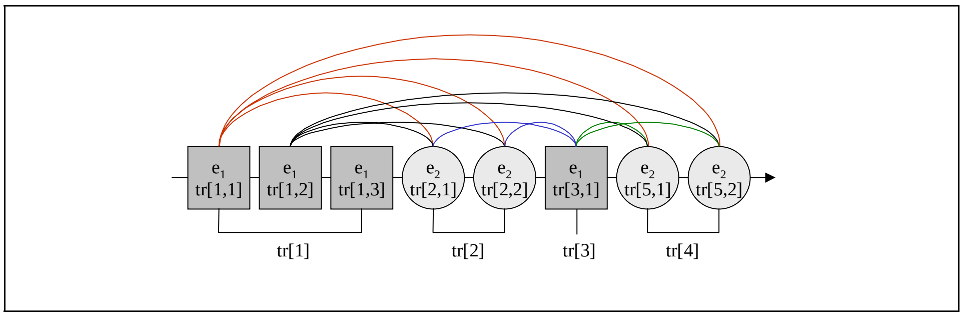

Fig. 11. Conjunction $p_5:=(e_1,e_2)$ with $z_{min}=1$, $z_{max}= 2$, $w_{min}=1$, $w_{max}=2$ as well as $P_{xy}:(x<=y)$

Fig. 11, however, shows how the value of P_{xy} selects all pairs which are allowed by $z_{min}=1$, $z_{max}=2$ and $w_{min}=1$, $w_{max}=2$. From each trace view, the two first duplicates of e_1 are combined with the two first duplicates of e_2 or vice versa. An exception is *tr*[3], which contains a single instance of class E_1 only.

Sequence For profiles p_1 and p_2 as well as event $e_1, e_2 \in \mathbb{E}$ holds:

$$(p_1; p_2)_t \prec (e_1, e_2) := \{p_1 \prec e_1 \ \wedge\ p_2 \prec e_2,\ t(e_2) \in (t(e_1), t(e_1 + t)]\}$$

$$\begin{aligned}(p_1; p_2)_t(tr) \prec \{(tr[2x-1,z],\ tr[2y,w]) \mid\ & \\ & (tr[2x-1,z],\ tr[2y,w]) \in tr(e_1,e_2) \\ & \wedge (p_1; p_2)_t \prec (tr[x,z],\ tr[y,w]) \\ & \wedge x \in [1,\infty) \wedge y \in [1,\infty) \\ & \wedge w \in [w_{\min}, w_{\max}] \wedge z \in [w_{\min}, w_{\max}] \\ & \wedge\ P_{xy}\}\end{aligned}$$

The set of event pairs of a trace *tr*, which matches the sequence of p_1 and p_2, consists of the tuples (e_1,e_2) that match (p_1;p_2) and for which the events e_1 or e_2 occur in certain places of the respective trace views. That means $z\in[z_{min},z_{max}]$ determines those duplicates to be chosen for $e_1=tr[2x\text{-}1,z]$

whereas $w \in [w_{min}, w_{max}]$ determines those for $e_2=tr[2y,w]$. P_{xy} specifies as condition the relation between x and y. This conveys which pairs are chosen for (p_1,p_2)[2]. The following example illustrates the sequence (Fig. 12).

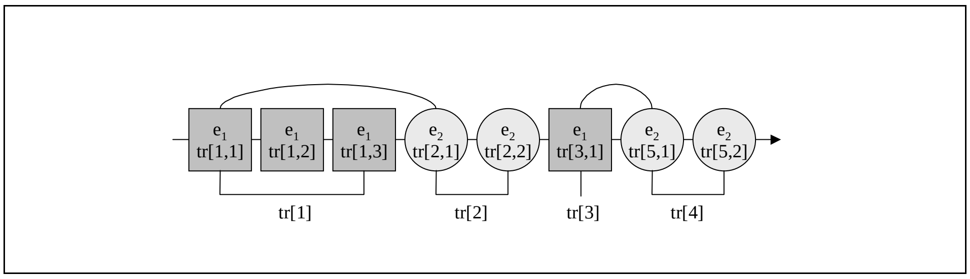

Fig. 12. Sequence $p_4{:=}(e_1;e_2)$ with $z_{min}=z_{max}= 1$, $w_{min}=w_{max}=1$ as well as $P_{xy}{:}(x=y)$

In Fig. 12, P_{xy} specifies that for the sequence we are considering unique pairs only, i.e. each event instance may take part in the sequence only once. $z_{min}=z_{max}=1$ as well as $w_{min}=w_{max}=1$ define that for each trace view, we are considering the first duplicate only.

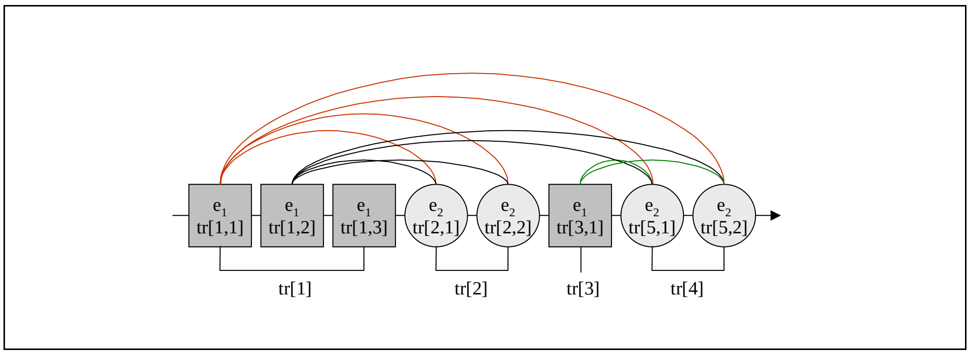

Fig. 13. Sequence $p_5{:=}(e_1;e_2)$, $z_{min}=1$, $z_{max}= 2$, $w_{min}=1$, $w_{max}=2$ and $P_{xy}{:}(x<=y)$

Fig. 13 shows an example of the sequence in which P_{xy} selects all pairs which are allowed by $z_{min}=1$, $z_{max}=2$ and $w_{min}=1$, $w_{max}=2$. So, for each odd trace view the two first duplicates of class E_1 are selected. They are selected together with the two first duplicates of class E_2 of each even trace view. A precondition is that the even trace view is located after the odd trace view, though this does not necessarily have to be directly after it.

Selection For a profile p, $\mathrm{E} \subset \mathbb{E}$ is the set of all events e which match p:

$$\mathrm{E} = \{tr[x,z] \mid x \in [1] \wedge z \in [z_{\min}, z_{\max}] \wedge tr[x,z] \in tr(e)\}$$

For $i \in \mathbb{N} \wedge i \leq |\mathrm{E}|$ we have:

$$p^{[i]}(tr) \prec e := \{tr[x,z] \mid z_{\min} = z_{\max} = i \wedge tr[x,z] \in E\}$$

From the trace tr the i^{th} event is selected (the presentation of the selection is based on [21]).

[2] As we have mentioned at the beginning of the section, we presume that the first occurrence of an event instance is always e_1. For the implementation, in which this is not always given, the definition has to be implemented in both directions. Then, we would have to consider $(tr[2x,z], tr[2y+1,w])$.

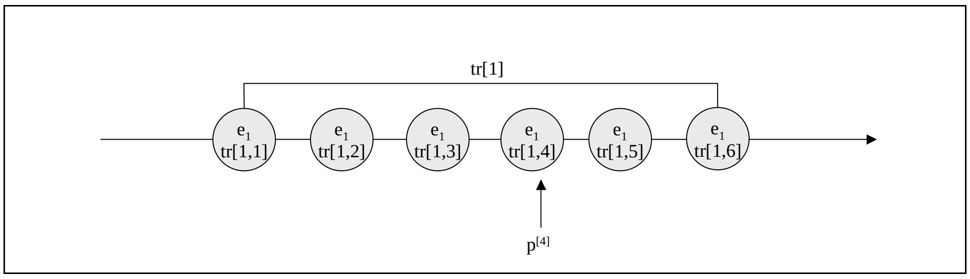

Fig. 14. Selection $p:e_1^{[4]}$

Parameters The parameters used in the definitions of the event operators can specify different values. Exemplary, in Table 11, we give an overview of the values which are most common for the duplicate parameter *z* and *w*. These two parameters describe the event instance selection, i.e. which events of each duplicate group will be selected:

Table 11. Duplicate parameter values

Selected event instances	z	w
first event	$z_{min}=z_{max}=1$	$w_{min}=w_{max}=1$
i^th event	$z_{min}=z_{max}=i$	$w_{min}=w_{max}=i$
last event	$z_{min}=z_{max}=length(tr_{ant})$	$w_{min}=w_{max}=m$
all events	$z_{min}=1$, $z_{max}=length(tr_{ant})$	$w_{min}=1$, $w_{max}=m$

Another parameter is the selection parameter. It specifies which pairs of event instances will be combined with each other. This expresses the consumption mode of the event operators with parameters.

Table 12. Selection parameter values

Selection	P_{xy}
unique pairs	P_{xy}: $x=y$
all pairs	P_{xy}: $x\leq y$

The value “unique pairs” signifies a consumption mode of “delete” and “all pairs” denotes “keep”, i.e. event instances can be filtered several times.

4.3 Classification

Both publications [26], [44] regarding the description of composite events or the definition of profiles which are fulfilled by these events show similarities in some areas. Nevertheless, they are not fully congruent since Unland and Zimmer subdivide the describing parameters further.

Both, Hinze and Voisard as well as Unland and Zimmer use the concept of composite events: The former use it in the context of event notification systems. There profiles filter composite events. The latter are discussing event-condition-action-rules in respect to active databases. Within such a rule an event is triggered by another event. If an event matches the specification of its event type, the condition is checked and, if applicable, the action is executed. The concepts for the description, nevertheless, remain the same in both cases.

Both publications employ as operators conjunction, disjunction, sequence and negation. However, the selection is only offered by Hinze and Voisard and the simultaneity by Unland and Zimmer. As in the following sections we are using the event algebra suggested by Hinze and

Voisard, we define the simultaneity in this technical report. It follows the event algebra introduced by Hinze and Voisard.

Simultaneity For profiles p_1 and p_2 as well as events $e_1,e_2 \in E$ holds:

$$(p_1 : p_2)_t \prec (e_1, e_2) \;:=\; \{p_1 \prec e_1 \;\wedge\; p_2 \prec e_2,\; t(e_1) = t(e_2)\}$$

$$\begin{aligned}(p_1 : p_2)_t(tr) \prec \{(tr[2x-1, z],\; tr[2y, w]) \mid\; & \\ & (tr[2x-1, z],\; tr[2y, w]) \in tr(e_1, e_2) \\ & \wedge (p_1 : p_2)_t \prec (tr[x, z],\; tr[y, w]) \\ & \wedge x \in [1, \infty) \wedge y \in [1, \infty) \\ & \wedge w \in [w_{\min}, w_{\max}] \wedge z \in [w_{\min}, w_{\max}] \\ & \wedge P_{xy}\}\end{aligned}$$

The set of event pairs of trace *tr* which match the simultaneity of p_1 and p_2 consists of the tuples (e_1,e_2) which match $(p_1{:}p_2)$ and for which events e_1 and e_2 occur at a certain place of the respective trace view. This means that $z \in [z_{min},z_{max}]$ determines the duplicates which have to be selected for $e_1=tr[2x-1,z]$, and $w \in [w_{min},w_{max}]$ determines the duplicates for $e_2=tr[2y,w]$. P_{xy} specifies the relation between *x* and *y*. This yields which pairs have to be selected for (p_1,p_2).

Both, Unland and Zimmer as well as Hinze and Voisard offer the idea of a timeframe for operators. This is required for the negation operator and the exclusive disjunction (if any system defines the disjunction operator in that way). For the actual implementation of the simultaneity operator, we can use a timeframe with value ε. This is due to the fact that because of time differences between systems, a "real" simultaneity of two events is not detectable. Since this work analyses concepts underlying the formation of transformations, we chose a theoretical approach for the simultaneity. For the evaluation of other operators, a timeframe is helpful but not required.

The evaluation strategies regarding incoming event instances are handled differently by the authors - except for the consumption mode. This is represented by Unland and Zimmer by the values "shared", "exclusive parameter" and "exclusive". Hinze and Voisard have summarised these values by "all pairs" and "unique pairs". In our classification, we use the following notions: "keep" – this corresponds to the consumption mode of "shared" for Unland and Zimmer and "all pairs" for Hinze and Voisard; "delete" corresponds to "exclusive parameter" for Unland and Zimmer as well as "delete & reapply", which corresponds to the value of "exclusive" for Unland and Zimmer and the value of "unique pairs" for Hinze and Voisard.

The traversion mode described by Unland and Zimmer is theoretically relevant for the evaluation since it influences the result. Nevertheless, in none of the described systems it is specified. However, we can state that implicitly the traversion mode is always from left to right. This is due to the fact that all systems are using the temporal order for their evaluation. So, in reality we never have to deal with a traversion mode of right to left. Therefore, we do not include the traversion mode into our classification.

Furthermore, Unland's concurrency mode with the values overlapping and non-overlapping does not exist in Hinze and Voisard's work. This mode is rarely specified in real systems. Moreover, it is dependent on the values of the consumption mode (cf. Table 9). Thus, it is not suitable for the classification of profile definition languages.

What is more is that the coupling mode of Unland and Zimmer does not necessarily have to be included in our classification scheme. This mode refers to the characteristic whether composite events may be interrupted or not ("continuous" or "„non-continuous"). We do not need to include this mode since it can also be expressed by the negation operator and the use of wildcards.

Unland and Zimmer's concept of parameter selection is realised by Hinze and Voisard with the help of the duplicate handling. For example both can select the first, last and i^{th} event instance or the corresponding duplicate of an event. These values are the most common ones of the systems analysed in Section 3. Additionally, we add a category "detailed" as there are a few systems which offer a duplicate or parameter selection which is even more sophisticated. It is possible to express these sophisticated parameters with the help of the concepts of both publications, but we summarise them and neglect them for our transformations, as they are so rare.

The classification system for profile definition languages we can derive from our previous discussion follows the structure shown in Table 13:

Table 13. Classification scheme

Composite events
Operators: conjunction, disjunction, sequence, negation, simultaneity, selection
Timeframe
Consumption: keep, delete, delete & reapply
Duplicate handling: first, last, all, detailed

	Composite events	Operators						Timeframe	Consumption mode			Duplicate handling			
		Conjunction	Disjunction	Sequence	Negation	Simultaneity	Selection		keep	delete	delete & reapply	first	last	all	detailed
Facility management	x	x	x	x	x			x							
Hermes	No														
PLAN	X	x	x		x			x				x			
CompAS language	X	x	x	x	x		x	x	x	x	x	x	x	x	x
Cobea	X	x	x	x	x			x	x					x	
Conquer	X	x	x	x	x	x		x							
Corba notification service	No														
Elvin	No														
Eve	X	x	x	x	x	x		x			x	x			x
Regis/Darwin (language GEM)	X	x	x	x	x			x		x		x			
Keryx	No														
OpenCQ	X	x	x	x	x	x		x							
READY	x	x	x	x	x							x	x	x	x
Rebeca	x	x	x	x	x						x				
Salamander	No														
Samos	x	x	x	x	x			x			x	x			x
Sentinel (language Snoop)	x	x	x	x				x		x	x	x	x		
Siena	x			x						x		x			
Yeast	x	x	x	x				x				x		x	
Active house	x	x	x	x	x							x			

Table 14. Analysis of profile definition languages of several systems and applications

4.4 Language Groups

Based on the observations from our comparative study of languages in the previous section, we identify language groups (types) of filter languages for event-based systems. This is the final step required for the identification of typical event patterns and groups of filter languages for event notification services.

These language groups form the basis for the design of the meta-service for event notification and event-based communication. In the next section, we address the finding of rules for profile transformations between these language groups. Parameters for consumption mode and duplicate handling are very rarely explicitly described in the literature. For this reason, we did not include the parameters in the definition of groups – they will be considered separately. Thus, the languages are classified into groups based on their support for time frames and on their support for pattern operators.

Table 15. Groups of filter languages

Time-frame-less composite events	**Time-framed composite events**
CE: Simple composite events ***(conjunction, disjunction, negation)***	**TCE**: Simple time-framed composite events ***(conjunction, disjunction, sequence)***
SCE: Sophisticated composite events ***(CE** and **sequence)***	**OTCE**: Ordinary time-framed composite events ***(TCE** and **negation)***
	STCE: Sophisticated time-framed composite events ***(OTCE** and **simultaneity)***

We define five groups as shown in Table 15.. There are two groups without time frame support: CEs support conjunction, disjunction and negation; a group member is PLAN. SCEs support conjunction, disjunction, negation and sequences. Members are READY, Rebeca and Active House (CEA).

There are three groups with time frame support: TCE offer conjunction, disjunction and sequence. Members of this group are Yeast and Sentinel (language Snoop). The OTCEs support conjunction, disjunction, sequence and negation; members of this group are Samos, Cobea and GEM. STCE offer conjunction, disjunction, sequence, negation and simultaneity. Members of this group are Eve, Conquer and OpenCQ. The disequilibrium of the group assignment of negation and sequence is due to the different effect of time frames on the operators.

4.5 Summary of Findings Regarding a Classification of Filter Languages

The three steps of analysing profile languages presented in this section are our answer to the question of how to identify typical event patterns and language groups. Firstly (Sections 4.1 and 4.2), we analysed typical patterns for composite events in profile definition languages. Secondly (Section 4.3), we compared the profile definition languages based on a classification schema. Thirdly (Section 4.4), we used the classification to identify typical groups of profile definition languages. The findings of this section shall serve as a foundation for answering the problem of finding transformation rules between languages from different groups. This problem is addressed in the next section.

5 Transformations

The previous section presented our three steps towards a classification of filter languages into language groups. We now address the problem of translating filter expressions between languages that use different operators and semantics. The answer to this problem shall provide a set of transformation rules that form the core of the proposed Meta-ENS for integrating heterogeneous event notification services. Here, we therefore especially consider the challenge of translating a filter expression of the meta-service into the target language of other systems.

5.1 Transformation Methodology

For each language group, we introduce transformation rules for translating filter expressions defined at the Meta-ENS into equivalent filter expressions using a language of the group. As can be derived from the group definitions, a simple translation of filter expressions between groups is not possible. Instead, for different semantic concepts in two distinct groups, we have to find expressions that are semantically close. Additionally, auxiliary profiles and post-filtering may be required.

Profile transformation If a certain operator does not exist in one language, a transcription expression has to be used. These transcriptions may be more or less expressive than the source expression. Therefore, we define four types of transformations: equivalent, positive, negative and transferring transformation. We denote these transformations with the arrow-notation that is shown in Table 16.

Table 16. Types of transformations

Transformation	**Notation**
Equivalent transformation	$\longleftrightarrow$
Positive transformation	$\xrightarrow{+}$
Negative transformation	$\xrightarrow{-}$
Transferring transformation	$\xleftrightarrow{\#}$

It is an extension of the notation used for Boolean transformations [11]. Equivalent transformations lead to expressions that have identical result sets. Positive transformations result in expressions that are less selective than the original - potentially creating larger result sets; negative transformations result in more selective expressions compared to the original filter expression (creating smaller result sets). Transferring transformations use post-filtering and auxiliary profiles.

Post-filtering and auxiliary profiles For the considered transformations between language groups, not all of the original operations can be expressed in the languages of less powerful groups. In order to use weaker systems in cooperation with stronger ones, auxiliary profiles (i.e. additional filter expressions) have to be defined at the services. Filter results are delivered to the stronger system, which then needs to perform additional simple filter operations (post-filtering).

Notification transformation Differing from query transformation, the result set obtained in an event notification service is not simply a set of tuples or documents. For event notification services, the result reflects the filter expression, i.e. the temporal connection between the events is reported. If for two communicating systems, the less expressive system receives a message from a more expressive one, the notification might not be comprehensible to the less expressive filter language. Lets consider the following example:

Consider two systems A and B denoted in Fig. 15, where the filter language of A supports only sequences and disjunctions. The filter language of B supports only conjunctions. These systems cannot cooperate directly, since their set of filter operators are disjoint. In order to cooperate, system B defines a profile P_B at the Meta-ENS ($P_B = (E_1,E_2)_t$). The Meta-ENS transforms this expression into a profile P_{Meta} that is defined at system A: $P_{Meta} = ((E_1;E_2)_t|(E_2;E_1)_t)_t$ with $P_{Meta} \longleftrightarrow P_B$. When system A sends a notification $N_{Meta} = ((e_1;\ e_2)_t|(e_2;\ e_1)_t)_t$ to the Meta-ENS, system B is notified by the transformed message $N_B = (e_1,e_2)_t$. Thus, not only the filter expressions have to be transformed for the cooperation but also the notifications.

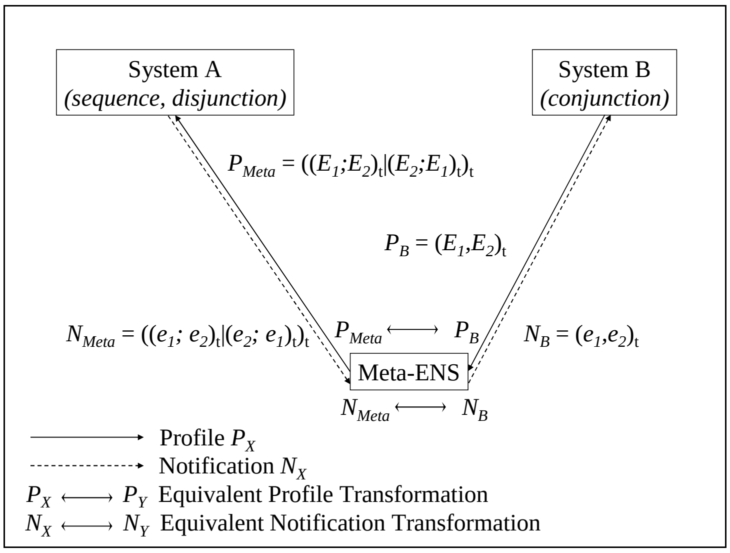
System A
(sequence, disjunction)
System B
(conjunction)
$P_{Meta} = ((E_1;E_2)_t|(E_2;E_1)_t)_t$
$P_B = (E_1,E_2)_t$
$N_{Meta} = ((e_1;\ e_2)_t|(e_2;\ e_1)_t)_t$
$P_{Meta} \longleftrightarrow P_B$
$N_B = (e_1,e_2)_t$
Meta-ENS
$N_{Meta} \longleftrightarrow N_B$
Profile P_X
Notification N_X
$P_X \longleftrightarrow P_Y$ Equivalent Profile Transformation
$N_X \longleftrightarrow N_Y$ Equivalent Notification Transformation

Fig. 15. Profile and notification transformation

The contributions of this section are a set of profile transformation rules for the interaction of the meta-service with other event notification services, auxiliary profile definitions and rules for post-filtering, and notification transformation rules. In this section, firstly we introduce the transformations for composite operators together with auxiliary profiles and post-filtering. Secondly, we define transformation rules for event pattern parameters which form the basic building block for our Meta-ENS.

5.2 Profile Transformation of Composite Operators

This section defines the transformation rules for composite operators. The rules are presented for the transformation of filter expressions defined at the Meta-ENS towards expressions of a target system within a given group (as identified in Section 4.4). We assume the Meta-ENS supports all concepts and event patters introduced in this technical report. We now iterate through the five target groups and show the necessary transformations.

Simple time-frame-less composite events (CE) For the members of this language group, for most operators we have to differentiate between two cases: When a profile is coming from the Meta-ENS, a timeframe may be given or not.

Conjunction, disjunction and negation can be mapped almost identically. However, the timeframe of the event algebra has to be set to ∞ in order to gain an equivalent transformation. This is due to the fact that timeframes do not exist in this language group. Nevertheless, if a timeframe is given, it will be lost when transforming into CE. Since the simultaneity does not exist in this language group, we have to simulate it with the help of the transferring transformation. For each $i \in \mathbb{N}$ of a selection $e_x^{[i]}$, we have to compose an own transformation rule. For an overview refer to Table 17.

Table 17. Transformation between CE and Meta-ENS

Operator	CE		Meta-ENS
Conjunction	(E_x,E_y)	$\longleftrightarrow$	$(E_x,E_y)_\infty$
	(E_x,E_y)	$\xleftarrow{+}$	$(E_x,E_y)_t$
Disjunction	$(E_x\|E_y)$	$\longleftrightarrow$	$(E_x\|E_y)_\infty$
	$(E_x\|E_y)$	$\xleftarrow{+}$	$(E_x\|E_y)_t$
Sequence	(E_x,E_y)	$\overset{\#}{\longleftrightarrow}$	$(E_x;E_y)_\infty$, $t(N(e_x))<t(N(e_y))$
	(E_x,E_y)	$\xleftarrow{+}$	$(E_x;E_y)_t$
Negation	$\overline{(E_x)}$ [3]	$\longleftrightarrow$	$\overline{(E_x)}_\infty$
	$\overline{(E_x)}$	$\xleftarrow{-}$	$\overline{(E_x)}_t$
Simultaneity	(E_x,E_y)	$\overset{\#}{\longleftrightarrow}$	$(E_x : E_y)_t$, $t(\mathcal{N}(e_x)) = t(\mathcal{N}(e_y))$
	(E_x,E_y)	$\xleftarrow{+}$	$(E_x : E_y)_t$
Selection	$((E_x)_t, \overline{(E_x,E_x)_t})_t$,	$\longleftrightarrow$	$E_x^{[1]}$
	$((E_x,E_x)_t, \overline{((E_x,E_x)_t,E_x)_t})_t$	$\longleftrightarrow$	$E_x^{[2]}$
	$\vdots$	$\longleftrightarrow$	$\vdots$

Sophisticated time-frame-less composite events (SCE) Conjunction, disjunction, negation and selection can be mapped in the same way as for the CE (simple timeframe-less composite events). The sequence is transformed analogously to conjunction and disjunction. The simultaneity is mapped by a combination of conjunction, sequence and negation. Please refer to Table 18.

Table 18. Transformation between SCE and Meta-ENS

Operator	SCE		Meta-ENS
Conjunction	(E_x,E_y)	$\longleftrightarrow$	$(E_x,E_y)_\infty$
	(E_x,E_y)	$\xleftarrow{+}$	$(E_x,E_y)_t$
Disjunction	$(E_x \mid E_y)$	$\longleftrightarrow$	$(E_x \mid E_y)_\infty$
	$(E_x \mid E_y)$	$\xleftarrow{+}$	$(E_x \mid E_y)_t$
Sequence	$(E_x;E_y)$	$\longleftrightarrow$	$(E_x;E_y)_\infty$
	$(E_x;E_y)$	$\xleftarrow{+}$	$(E_x;E_y)_t$

[3] Systems belonging to this group internally realise the negation by other means than a timeframe.

Negation	$\overline{(E_x)}$	$\longleftrightarrow$	$\overline{(E_x)}_\infty$
	$\overline{(E_x)}$	$\xleftarrow{-}$	$\overline{(E_x)}_t$
Simultaneity	$((E_x,E_y),((\overline{E_x;E_y}),(\overline{E_y;E_x})))$	$\longleftrightarrow$	$(E_x:E_y)_\infty$
	$((E_x,E_y),((\overline{E_x;E_y}),(\overline{E_y;E_x})))$	$\xleftarrow{+}$	$(E_x:E_y)_t$
Selection	$((E_x)_t,\overline{(E_x,E_x)_t})_t$	$\longleftrightarrow$	$E_x^{[1]}$
	$((E_x,E_x)_t,\overline{((E_x,E_x)_t,E_x)_t})_t$	$\longleftrightarrow$	$E_x^{[2]}$
	$\vdots$	$\longleftrightarrow$	$\vdots$

Simple time-framed composite events (TCE) Conjunction, disjunction and sequence can be transformed without any problems. The simultaneity can be constructed by using the conjunction, sequence and negation. Negation and selection do not exist within this language group, so their filtering has to be moved to the more powerful system using the transferring transformation. For the conjunction, disjunction and sequence, the timeframe *t* can be set to ∞ whenever required. Nevertheless, this will not be the case if the transformation from TCE is undertaken directly to our Meta-ENS. An overview can be found in Table 19.

Table 19. Transformation between TCE and Meta-ENS

Operator	TCE		Meta-ENS
Conjunction	$(E_x,E_y)_t$	$\longleftrightarrow$	$(E_x,E_y)_t$
Disjunction	$(E_x \mid E_y)_t$	$\longleftrightarrow$	$(E_x \mid E_y)_t$
Sequence	$(E_x;E_y)_t$	$\longleftrightarrow$	$(E_x;E_y)_t$
Negation	$(E_x)_t$	$\overset{\#}{\longleftrightarrow}$	$\overline{(E_x)}_t$, $\overline{(\mathcal{N}(e_x))}_t$
Simultaneity	$((E_x,E_y)_t,((\overline{E_x;E_y})_t,(\overline{E_y;E_x})_t))$	$\longleftrightarrow$	$(E_x:E_y)_t$
Selection	$(E_x)_t$	$\overset{\#}{\longleftrightarrow}$	$E_x^{[i]}$, $(\mathcal{N}(e_x))^{[i]}$

Ordinary time-framed composite events (OTCE) Conjunction, disjunction and simultaneity are transformed in an analogous way to the same operators of group TCE (simple time-framed composite events). The negation can be directly transformed. For each $i \in \mathbb{N}$ of the selection $e_x^{[i]}$, we have to build an own transformation rule. For conjunction, disjunction and sequence, the timeframe *t* can be set to ∞ whenever required. Nevertheless, this will not be the case if the transformation from the OTCE is undertaken directly to our Meta-ENS. For an overview, refer to Table 20.

Table 20. Transformation between OTCE and Meta-ENS

Operator	OTCE		Meta-ENS
Conjunction	$(E_x,E_y)_t$	$\longleftrightarrow$	$(E_x,E_y)_t$
Disjunction	$(E_x \mid E_y)_t$	$\longleftrightarrow$	$(E_x \mid E_y)_t$
Sequence	$(E_x;E_y)_t$	$\longleftrightarrow$	$(E_x;E_y)_t$
Negation	$\overline{(E_x)}_t$	$\longleftrightarrow$	$\overline{(E_x)}_t$
Simultaneity	$((E_x,E_y),((\overline{E_x;E_y}),(\overline{E_y;E_x})))_t$	$\longleftrightarrow$	$(E_x:E_y)_t$
Selection	$((E_x)_t,\overline{(E_x,E_x)_t})_t$	$\longleftrightarrow$	$E_x^{[1]}$

$((E_x, E_x)_t, (\overline{(E_x, E_x)_t, E_x})_t)_t$	$\longleftrightarrow$	$E_x^{[2]}$
$\vdots$	$\longleftrightarrow$	$\vdots$

Sophisticated time-framed composite events (STCE) The operators of this language group can be transformed almost directly into the event algebra since as most powerful group, it possesses almost all semantic concepts. For the selection, we simply have to develop a transformation rule for each $i \in \mathbb{N}$ of $e_x^{[i]}$. For conjunction, disjunction and sequence, the timeframe *t* can be set to ∞ whenever required. Nevertheless, this will not be the case if the transformation from the OTCE is undertaken directly to our Meta-ENS. An overview is given in Table 21.

Table 21. Transformation between OTCE and Meta-ENS

Operator	STCE		Meta-ENS
Conjunction	$(E_x, E_y)_t$	$\longleftrightarrow$	$(E_x, E_y)_t$
Disjunction	$(E_x \mid E_y)_t$	$\longleftrightarrow$	$(E_x \mid E_y)_t$
Sequence	$(E_x ; E_y)_t$	$\longleftrightarrow$	$(E_x ; E_y)_t$
Negation	$\overline{(E_x)}_t$	$\longleftrightarrow$	$\overline{(E_x)}_t$
Simultaneity	$(E_x : E_y)_t$	$\longleftrightarrow$	$(E_x : E_y)_t$
Selection	$((E_x)_t, \overline{(E_x, E_x)_t})_t$	$\longleftrightarrow$	$E_x^{[1]}$
	$((E_x, E_x)_t, (\overline{(E_x, E_x)_t, E_x})_t)_t$	$\longleftrightarrow$	$E_x^{[2]}$
	$\vdots$	$\longleftrightarrow$	$\vdots$

5.3 Transformation of Operator Parameters

The group definitions given in Section 4.4 abstracted from the parameters of consumption mode and duplicate handling strategy since these parameters are rarely explicitly supported in the considered systems. In this section, we show the influence of considering parameter transformations on operator transformations (as introduced in the previous section). This previous section presented our answer to the problem of translating filter expressions between languages that use different operators and semantics. We provided a set of transformation rules that form the core of the proposed Meta-ENS for integrating heterogeneous event notification services. The transformation rules presented here have also been implemented in a prototype transformation component that can be used with any given event notification service.

In this section, we will exemplarily introduce parameterised transformations. We will do this for a system of the Universität Zürich that has been described in detail in research publications. The system, Samos, has the value “first” for the duplicate parameter and can also chose the i*th* duplicate. For the consumption mode, it uses “delete & reapply”.

Additionally to these example transformations, we give an overview of what possibilities for parameterised transformations exist at all. This overview takes into account which parameter values of a system A can be transformed into which parameter values of a system B. Generally, we can state that the consumption mode presented by Unland and Zimmer [44] with a value of “delete” can be mapped to the selection parameter of Hinze und Voisard [26] by using a value of “unique pairs” with $P_{xy} : x = y$. For the value “keep”, it is transformed by the selection parameter with the value “all pairs” with $P_{xy} : x \le y$.

Exemplary parameter transformation for Samos Exemplarily for the transformation of the parameters, we give the transformation of Samos with the value of “delete & reapply” for the consumption mode and a duplicate parameter of ”first”. In the Samos system, the selection of the first duplicate is realised by the closure-constructor *. If we transfer this into the event algebra the duplicate parameter is represented by $(z_{\min} = 1) \wedge (z_{\max} = 1)$, $(w_{\min} = 1) \wedge (w_{\max} = 1)$ and the selection parameter is specified with the help of $P_{xy} : x = y$.

In the following, we present our transformations from Samos (left-hand-side of the transformation, red) to the Meta-ENS (right-hand-side of the transformation) for the different operators.

Conjunction

$$(*(E1, E2))\ IN\ [start\_time,\ end\_time]$$
$$\longleftrightarrow$$
$$(p_1, p_2)\ (tr)\ with\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge (w_{\min} = 1) \wedge (w_{\max} = 1) \wedge P_{xy} : x = y$$
$$\wedge p_1 \prec E1 \wedge p_2 \prec E2 \wedge t = [end\_time - start\_time]$$

Disjunction In Samos, it is possible to specify a timeframe for the disjunction. In the definition of the parameterised event algebra, this is not possible. This is why, we require two different kinds of transformations depending on the timeframe given in Samos.

$$(*(E1 \mid E2))\ IN\ [start\_time,\ end\_time]$$
$$\xrightarrow{+}$$
$$(p_1 \mid p_2)\ (tr)\ with\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge (w_{\min} = 1) \wedge (w_{\max} = 1) \wedge P_{xy} : x = y$$
$$\wedge p_1 \prec E1 \wedge p_2 \prec E2$$

and

$$(*(E1 \mid E2))\ IN\ [\infty]$$
$$\longleftrightarrow$$
$$(p_1 \mid p_2)\ (tr)\ with\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge (w_{\min} = 1) \wedge (w_{\max} = 1) \wedge P_{xy} : x = y$$
$$\wedge p_1 \prec E1 \wedge p_2 \prec E2$$

Sequence

$$(*(E1; E2))\ IN\ [start\_time,\ end\_time]$$
$$\longleftrightarrow$$
$$(p_1; p_2)\ (tr)\ with\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge (w_{\min} = 1) \wedge (w_{\max} = 1) \wedge P_{xy} : x = y$$
$$\wedge p_1 \prec E1 \wedge p_2 \prec E2 \wedge t = [end\_time - start\_time]$$

Negation For the negation operator, duplicate handling and consumption mode are irrelevant. It is not necessary to determine which duplicate of an event is selected or if it will be consumed or is available for refiltering after being used for the event composition. This is due to the fact that we are considering the absence of an event and not its occurrence.

$$
\begin{aligned}
&(NOT\ \ E)\ IN\ [start\_time,\ end\_time]\\
&\longleftrightarrow\\
&\overline{(p_1)}_{t1}\ (tr)\ with\ p_1 \prec E \wedge t_1 = [end\_time - start\_time]
\end{aligned}
$$

Simultaneity Samos does not offer this operator. Thus, we have to construct the simultaneity by using the conjunction, sequence and negation.

$$
\begin{aligned}
&((*(((*(E1,E2))\ IN\ [start\_time,\ end\_time]),\\
&\qquad ((*(\qquad ((NOT\ \ ((*(E1;E2))\ IN\ [start\_time,\ end\_time]))\\
&\qquad\qquad\qquad\qquad IN\ [start\_time,\ end\_time]),\\
&\qquad\qquad ((NOT\ \ ((*(E2;E1))\ IN\ [start\_time,\ end\_time]))\\
&\qquad\qquad\qquad\qquad IN\ [start\_time,\ end\_time])\\
&\qquad))\ IN\ [start\_time,\ end\_time])\\
&))\ IN\ [start\_time,\ end\_time])\\
&\longleftrightarrow\\
&(p_1 : p_2)\ (tr),\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge (\mathrm{w}_{\min} = 1) \wedge (\mathrm{w}_{\max} = 1) \wedge P_{xy} : x = y\\
&\qquad\qquad \wedge p_1 \prec E1 \wedge p_2 \prec E2 \wedge t = [end\_time - start\_time]
\end{aligned}
$$

Selection For each $i \in \mathbb{N}$ of the selection $p^{[i]}$, we have to specify an own transformation rule. Here, we exemplarily give the rule for *i=1*:

$$
\begin{aligned}
&(*(\\
&\qquad ((*E1)\ IN\ [start\_time,\ end\_time]),\\
&\qquad ((NOT\ \ ((*(E1,E1))\ IN\ [start\_time,\ end\_time]))\\
&\qquad\qquad\qquad IN\ [start\_time,\ end\_time])\\
&))\ IN\ [start\_time,\ end\_time]\\
&\longleftrightarrow p^{[1]}\ (tr)\ with\ (z_{\min} = 1) \wedge (z_{\max} = 1) \wedge p \prec E1
\end{aligned}
$$

Transferability of parameters The transformations of the event operators that have been discussed in Section 5.2 will now be extended by the duplicate and the selection parameter. In Table 22, you can find an overview of the transferability of all different values of these parameters into each other.

Table 22. Parameter transformations: duplicate handling and consumption mode parameter

first	**all**	⤡							
	unique	↖$^{+}$	⤡						
last	**all**	-	-	⤡					
	unique	-	-	↖$^{+}$	⤡				
all	**all**	↘$^{+}$	↘$^{+}$	↘$^{+}$	↘$^{+}$	⤡			
	unique	-	-	-	-	↖$^{+}$	⤡		
ith[4]	**all**	↘$^{+}$	↘$^{+}$	-	-	↖$^{+}$	-	⤡	
	unique	-	↘$^{+}$	-	-	↖$^{+}$	↖$^{+}$	↖$^{+}$	⤡
duplicate		**first**		**last**		**all**		**ith[4]**	
parameter	**selection**	**all**	**unique**	**all**	**unique**	**all**	**unique**	**all**	**unique**

For all combinations of values for the duplicate and selection parameter, we show their possibility of transforming them. The direction of the arrows presented indicates that a transformation is possible. For a direction to the left, the expression in a column can be transformed into the expression in a row. The other direction works analogously.

6 Implementation

Within the scope of this work, we have implemented a transformator which is a prototype of the meta-service that was mentioned as motivation for our work. The transformator realises transformations into the event algebra for some of the language groups and vice versa. It can be integrated into DAS [5]. There the transformator can be used to support communication between other systems as well as a mediator for different event notification systems translating different languages and concepts.

For the realisation of the transformator, we have used the programming language Prolog since it is based on logic concepts. We employed SWI-Prolog (Version 3.2.6), a free version of Prolog by the Universität Amsterdam. The basic structure of Prolog is based on facts, rules, clauses and questions. As data basis, a number of facts is defined. With these, it is possible to define rules which describe relations between arguments and predicates. This set of rules can be evaluated with the help of questions using variables. All arguments can be calculated by the use of the rules previously defined. A determination of an unknown we may undertake from both directions. This structure is similar to the formation of our transformations, which are defined in two directions.

6.1 Realisation of the Transformator

For the realisation of our transformator, we have considered three different aspects, which are described in the following:

[4] Here, we consider how powerful a language is and not the actual value of i.

Syntax mapping The construction of events has to be undertaken in a way they can be used by Prolog. For this, we had to find expressions for primitive and composite events, timeframes, timestamps and operators.

Correctness check The implementation does not only have to offer a realisation of the expressions of the event algebra but also provide the possibility to check them semantically. For example, the sequence $(e_1;e_2)$ shall be transformed only if the timestamp of e_1 is smaller than that of e_2. Otherwise, the transformation has to be discarded.

Transformation In the programme, we have exemplarily realised transformations for the language groups ordinary time-framed composite events, sophisticated time-framed composite events as well as simple timeframe-less composite events.

6.2 Functionality of the Programme

Presentation of events Primitive events are represented as tuples. The first element stands for the event name and the second realises the timestamp of the event. Event names are realised by the identifiers “e1” to “e9”. As an example, see the rule for primitive event e1:

```
pEvent([e1, Timestamp]):-   integer(Timestamp).
```

For checking the correctness, the given timeframe is tested for its membership in the integer domain.

The operators for the representation of composite events are divided into binary and unary operators. Both of them require a different presentation. Binary operators are conjunction, disjunction, sequence and simultaneity. They are realised as list containing four elements. The first element describes the operator, the second and third stand for the events that are taking part and the last element represents the timeframe. Within this timeframe, the composite event has to take place. As an example, please view the following sequence. For the correctness test, the events have to occur in the correct order within the given timeframe:

```
comEvent([sequence, Event1, Event2, Timeframe]):-
     sequence(Event1, Event2, Timeframe).

sequence(Event1, Event2, Timeframe):-event(Event1),
     event(Event2), isEarlier(Event1, Event2),
     withinTimeframe(Event1, Event2, Timeframe).
```

The unary operators are negation and selection. They are implemented as triples. The first element denotes the operator, the second stands for the event and the third for the timeframe or the parameter to be selected by the selection.

Supported operators Examples for the syntax of the unary operators, negation and selection, are given as triples as follows:

```
[negation, evt, 10]
```

Hereby, we describe the negation of event *evt*. This means that event *evt* has not occurred for 10 seconds. *evt* can be a primitive event as well as a composite event.

```
[selection, evt, 3]
```

The previous example describes a selection in our implementation. It selects the third duplicate of event *evt*.

The binary operator representing the disjunction can be utilised as follows:

```
[disjunction, evt1, evt2, 3]
```

The events *evt1* and *evt2* can denote primitive or composite events. As timeframe for our example, we have chosen three seconds. Analogously, we can use the binary operators, conjunction, sequence and simultaneity as `conjunction`, `sequence` and `simultaneity.` This uses the same syntax as a list with four elements.

Supported language groups The ordinary time-framed composite events, the sophisticated time-framed composite events and the simple timeframe-less composite events are represented in the implementation as group1, group2 and group3. An integration of further groups is easily possible as the transformations are specified in tables. The development of these tables is possible without changing the logic of the programme.

Exemplary use of programme In the following section, we are showing several short programme sequences.

```
?- event([sequence, [e1,3],[e2,5], 10]).

Yes

?- event([sequence, [e1,3], [e2,20], 10]).

No
```

In the first case, we have a sequence that occurs within the given timeframe. The second case shows an example where the event occurs after the given timeframe has ended. This is why, this event is discarded.

```
?- transformation([event-algebra, [simultanity,
    [e1,10], [e2,10], 40]], [group3, X]).


X = [conjunction, [e1, 10], [e2, 10]]

Yes
```

When transforming a simultaneity from the event algebra into the group of the simple timeframe-less composite events, we loose information. On the one hand, this group does not offer a simultaneity operator, and on the other hand it does not use timeframes. This is why, according to our transformation rules we have to transform it into the conjunction without a timeframe.

```
?- transformation([event-algebra, [sequence, [e1,4],
    [e3,6], 10]], X).


X = [group1, [sequence, [e1, 4], [e3, 6], 10]] ;

X = [group2, [sequence, [e1, 4], [e3, 6], 10]] ;

X = [group3, [conjunction, [e1, 4], [e3, 6]]]

Yes
```

Here, we are querying the programme to execute all possible sequences by the event algebra. Not only the ordinary but also the sophisticated time-framed composite events are able to directly map this transformation. The group of simple timeframe-less composite events have to transform into the conjunction without using a timeframe.

6.3 Integration into CompAS

The CompAS system has been implemented in Java. Nevertheless, our transformator can be integrated. This works with the help of the interface JIPL (Java Interface for Prolog) offered by k-Prolog [2]. Using this interface, it is possible to invoke Prolog-predicates directly from Java. This way, a composite event existing in the CompAS system can be transformed by calling the Prolog implementation. Furthermore, the CompAS system can be used directly by Prolog.

7 Conclusion and Outlook

In this section, we give an evaluation of our results and discuss a potential application of our research. Based on that, we give suggestions for future work. We conclude with a summary of the technical report.

7.1 Evaluation

This technical report is part of the research which has been undertaken in the project MediAS [24] at the Institut für Informatik at Freie Universität Berlin. Within the scope of this project, an event notification service has been developed with its different versions PrimAS, CompAS and DAS. This theoretical work is of use to the CompAS system and its distributed variant DAS as it forms the basis for a communication using differing profile definition languages.

In doing so, the transformations we have found require an hierarchical approach in their application in order to fully exploit the powerfulness of differing language groups. As indicated in the introduction, it is possible that we encounter loss of information when transforming composite events. An example is the transformation of the sequence $(e_3;e_5)$ into the conjunction $(e_3,e_5)_\infty$. If a publisher neglects temporal matters, we loose the information that event e_3 temporally occurred before event e_5. If we are using a sequence of systems, the one with the least expressive profile definition language will determine the degree of information. Therefore, it would be reasonable to request information from systems with more expressive profile definition languages whenever possible. Only if this is impossible, systems with less expressive profile definition languages should be used as publishers. The expressiveness of our language groups increases within the hierarchy presented in Table 23.

Table 23. Hierarchy of expressiveness

Simple time-frameless composite events
Sophisticated time-frameless composite events
Simple time-framed composite events
Ordinary time-framed composite events
Sophisticated time-framed composite events

The highest precision is offered by the group of the sophisticated time-framed composite events. The greatest inaccuracies occur if we use languages that do not utilize timeframes. This is due to the fact that timeframes cannot be reconstructed from any other information.

In some language groups, not all missing event operators are expressible by the use of existing event operators of the respective group. This is why, we have introduced the transferring transformation in order to make possible the use of systems belonging to these language groups. This kind of transformation is a good means for the representation of non-existent operators. Some components of an event in question are directly requested from the other system. Nevertheless, the actual filtering to gain a composite event is transferred to the CompAS system. The advantage of this is that all event operators can be simulated. The disadvantage is the higher load on the system to which the filtering is transferred.

When using the transformator, it is important to keep in mind that due to the lack of detail in the respective research publications, we do not know the values for the duplicate and selection parameters. This might cause inaccuracies concerning the evaluation of the incoming events. To target this problem, we propose the approach described in the following section.

7.2 Future Work

The following issues could still be covered within the general project matter. Nevertheless, they exceeded the scope of the thesis described in this technical report:

Practical analysis The analysis of different event notification services presented in this technical report could be extended by a practical analysis. This practical analysis could examine systems for those parameters which are not fully described in research publications. The lack of these parameters makes an evaluation of incoming events difficult. Mostly, authors neither describe what values they are using for the consumption mode nor how they handle the duplicate parameter. To target this problem, it would be a possibility to install the respective systems. This way, we are able to test them by defining appropriate profiles in order to find out how they deal with these parameters. These results could be used for the classification of groups and thus, the groups could be categorised more detailed.

Implementation of complete concept Our exemplary implementation of transformations for three language groups could be extended by adding the two remaining language groups.

Integration of transformator The transformator could be integrated into the CompAS system which has been extended in DAS [5] to a distributed event notification service supporting composite events.

Integration of parameters If the practical analysis (described above) has been undertaken, the consumption mode and duplicate parameter can be integrated into the transformator and the CompAS system. These parameters do not refer to the part of the transformator which has been realised in Prolog but concerns the filtering process of profiles within the CompAS system. For the integration, Table 22, describing the theoretical possibilities, can be used.

7.3 Summary

This technical report has presented answers to the following two research problems: Firstly, subscribers of heterogeneous event notifications services are forced to subscribe the same profile to a number of services using different filter languages. Secondly, composite events combining events from different providers that are handled by different services have to be identified by a subscriber-based post-filtering.

As a solution to these two problems, we proposed the detailed design of a Meta-Event Notification Service based on transformation rules. In particular, this technical report presented the following contributions: Firstly, we presented a survey of filter languages for event notification.

Secondly, we introduced a classification schema for profile definition languages. Thirdly, we identified five categories of profile languages. Fourthly, we proposed detailed transformation rules for translating profiles defined at the Meta-ENS into languages of systems from the five categories (and vice versa for notifications).

As proof of concept, we have implemented a transformation component for our proposed language transformations. The implementation was carried out using Prolog. The transformation component currently supports our operator transformations. The next version of the transformation component will also incorporate the proposed parameter transformation.

The next step in our research will see the close integration of the transformation component into our prototypical event notification system A-MediAS [13]. The transformation can be used for the role of a Meta-ENS in the communication with other event notification services (as providers) and for the mediation between event notification services (as providers and subscribers).

References

1. CORBA Notification Service Specification, OMG Document 00-06-20, Object Management Group, 2002.
2. Karakuri Logic Systems, 2002.
3. Belokosztolszki, A. The Active House Project, Opera Group, University of Cambridge Computer Laboratory, Cambridge, 2002.
4. Bittner, S. *Implementierung eines effizienten Filteralgorithmus für Benachrichtigungssysteme*. Thesis (Studienarbeit). Department of Computer Science, Freie Universität Berlin, Berlin, 2002.
5. Bittner, S. *Implementierung und Analyse eines verteilten Benachrichtigungsdienstes*. Thesis (Diplomarbeit). Department of Computer Science, Freie Universität Berlin, Berlin, 2003.
6. Brandt, S. and Kristensen, A. Web push as an Internet notification service *W3C Workshop on Push Technology*, Boston, USA, 1997.
7. Carzaniga, A., Rosenblum, D.R. and Wolf, A.L., Challenges for Distributed Event Services: Scalability vs. Expressiveness. in *In Proceedings of the 21st International Conference on Software Engineering (ISCE'99) Workshop on Engineering Distributed Objects (EDO'99)*, (Los Angeles, USA, 1999), 72-77.
8. Carzaniga, A., Rosenblum, D.S. and Wolf, A.L. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, *19* (3). 332-383, 2001.
9. Chakravarthy, S. and Mishra, D., Sentinel: An Object-Oriented DBMS with Event-Based Rules. in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD 1997)*, (Tucson, USA, 1997), 572-575.
10. Chakravarthy, S. and Mishra, D., Snoop: An Expressive Event Specification Language for Active Databases. Technical Report, UF-CIS-TR-93-007. University of Florida, Computer and Information Sciences Department, 1993.
11. Chang, C.-C.K., Garcia-Molina, H. and Paepcke, A. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM Transactions on Information Systems, 17* (1). 1-39, 1999.
12. Faensen, D., Faulstich, L., Schweppe, H., Hinze, A. and Steidinger, A., Hermes - A Notification Service for Digital Libraries. in *Proceedings of the ACM/IEEE Joint Conference on Digital Libaries (JCDL2001)*, (Roanoke, USA, 2001), 373-380.
13. Fiege, L., Mühl, G. and Gärtner, F.C., A Modular Approach to Building Event-Based Systems. in *Proceedings of the ACM Symposium on Applied Computing 2002 (SAC'02)*, (Madrid, Spain, 2002), 385-392.

14. Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T. and Segall, B., Augmenting the workaday world with Elvin. in *Proceedings of the 6th European Conference on Computer Supported Cooperative Work (ECSCW'99)*, (Copenhagen, Denmark, 1999), 431-451.

15. Gatziu, S. and Dittrich, K.R. SAMOS: an Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases, 15* (1-4). 23-26, 1992.

16. Geppert, A. and Tombros, D. Event-based Distributed Workflow Execution with EVE. *Proceeding of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*. 427-442, 1998.

17. Gruber, R.E., Krishnamurthy, B. and Panagos, E. The Architecture of the READY Event Notification Service *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS 1999) Workshop on Electronic Commerce and Web-Based Applications*, Austin, USA, 1999.

18. Gruber, R.E., Krishnamurthy, B. and Panagos, E. High-Level Constructs in the READY Event Notification System. *Proceedings of 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*. 195-202, 1998.

19. Hayton, R. *OASIS - An Open Architecture for Secure Interworking Services*. PhD Thesis. Computer Laboratory, Cambridge University, Cambridge, UK, 1996.

20. Heinrich, K., Lichtvision - Gesellschaft für Lichttechnik und Gebäudemanagement, Berlin, Germany, 2002.

21. Hinze, A., Freie Universität Berlin, Berlin, Germany, 2002.

22. Hinze, A., Does Alerting have Special Requirements for Query Languages? in *Tagungsband zum 13. GI-Workshop Grundlagen von Datenbanken (GvD 2001)*, (Gommern, Germany, 2001), 58-62.

23. Hinze, A. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD Thesis. Department of Computer Science, Freie Universität Berlin, Berlin, 2003.

24. Hinze, A., Bittner, S. and Jung, D. Project Description MediAS, Freie Universität Berlin, Berlin, Germany, 2002.

25. Hinze, A. and Faensen, D., A Unified Model of Internet Scale Alerting Services. in *Proceedings of the 5th International Computer Science Conference (ICSC'99)*, (Kowloon, Hong Kong, 1999), 284-293.

26. Hinze, A. and Voisard, A., Composite events in notification services with application to logistics support. in *Proceedings of the 9th International Symposium on Temporal Representation and Reasoning (TIME-2002)*, (Manchester, UK, 2002).

27. Hinze, A. and Voisard, A., A flexible parameter-dependent algebra for event notification services. Technical Report, tr-b-02-10. Freie Universität Berlin, Berlin, Germany, 2002.

28. Jung, D. *Benachrichtigungssysteme: Analyse und Transformation ausgewählter Profildefinitionssprachen*. Thesis (Staatsexamensarbeit). Department of Computer Science, Freie Universität Berlin, Berlin, Germany, 2002.

29. König, S. *Implementierung und Untersuchung eines parametergesteuerten Benachrichtigungssystems für kombinierte Events*. Diplomarbeit. Department of Computer Science, Freie Universität Berlin, Berlin, Germany, 2005.

30. Krishnamurthy, B. and Rosenblum, D.S. Yeast: A General Purpose Event-Action System. *IEEE Transactions on Software Engineering, 21* (10). 845-857, 1995.

31. Liebig, C., Cilia, M. and Buchmann, A.P., Event Composition in Time-dependent Distributed Systems. in *Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems (CoopIS'99)*, (Edinburgh, UK, 1999), 70-78.

32. Liu, L., Pu, C. and Tang, W. Continual queries for internet scale eventdriven information delivery. *IEEE Transactions on Knowledge and Data Engineering, Special issue on Web Technologies, 11* (4). 610-628, 1999.

33. Liu, L., Pu, C., Tang, W. and Han, W. CONQUER: A Continual Query System for Update Monitoring in the WWW. *International Journal of Computer Systems, Science, and Engineering, Special edition on Web Semantics*, 1999.

34. Ma, C. and Bacon, J., COBEA: A CORBA-Based Event Architecture. in *Proceedings of the USENIX Conference on Object-Oriented Technologies & Systems (COOTS'98)*, (Santa Fe, USA, 1998), 117-131.

35. Malan, G.R., Jahanian, F. and Subramanian, S., Salamander: A push-based distribution substrate for internet applications. in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'97)*, (Monterey, USA, 1997), 171-181.

36. Mansouri-Samani, M. and Sloman, M., GEM: A Generalised Event Moni-toring Language for Distributed Systems. in *Proceeding of the International Con-ference on Open Distributed Systems and Distributed Platforms (ICODP/ICDP'97)*, (Toronto, Canada, 1997), 96-108.

37. Mühl, G., Generic Constraints for Content-Based Publish-Subscribe. in *In Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS'01)*, (Trento, Italy, 2001), 211-225.

38. Naughton, J., DeWitt, D., Maier, D., Chen, J., Galanis, L., Tufte, K., Kang, J., Luo, Q., Prakash, N. and Tian, F. The Niagara Internet Query System. *IEEE Data Engineering Bulletin*, *24* (2). 27-33, 2001.

39. Pu, C. and Liu, L., Update Monitoring: The CQ project. in *Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications (WWCA'98)*, (Tsukuba, Japan, 1998), 396-411.

40. Segall, B., Elvin: Event notification service. Internal Report, DSTC, 1995.

41. Tombros, D., Geppert, A. and Dittrich, K.R., Semantics of Reactive Components in Event-Driven Workflow Execution. in *Proceedings of the 9th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, (Barcelona, Spain, 1997), 409-442.

42. Urban, S.D., Unruh, A., Martin, G. and Nodine, M., Expressing Composite Events in Infosleuth. Technical Report, MCC-INSL-131-98. Microelectronics and Computer Technology Corporation, 1998.

43. Wu, B. and Dube, K. PLAN: a Framework and Specification Language with an Event-Condition-Action (ECA) Mechanism for Clinical Test Request Protocols *34th Hawaii International Conference on System Science (HICSS-34)*, Maui, USA, 2001.

44. Zimmer, D. and Unland, R., The Formal Foundation of the Semantics of Complex Events in Active Database Management Systems. Technical Report, 22/1997. Cooperative Computing & Communication Laboratory, Paderborn, Germany, 1997.