Chapter 9

The A-MEDIAS System

One must learn by doing the thing,
for though you think you know it,
you have no certainty until you try.
Sophocles

In this chapter, we present our A-MEDIAS system – an adaptive integrating event notification system that realizes the theoretical design concepts introduced throughout this thesis. We illustrate how A-MEDIAS[1] implements our concepts for adaptive integrating event notification services.

Section 9.1 gives an overview of the A-MEDIAS system architecture. Successively, we present a more detailed view on the system. We start with the distributed A-MEDIAS system that consists of a network of servers. Within each server, the main component is the event notification component. Additionally, components for distributed access on events and profiles are provided. Here, We concentrate on specific implementation aspects of the event notification component.

In Section 9.2, we briefly describe the important classes of our A-MEDIAS implementation. The classes are presented as diagrams using the Unified Modelling Language (UML). Sections of selected class definitions are given in Appendix A.

Section 9.3 introduces the profile definition language of A-MEDIAS. In Chapter 5, we proposed our parameterized event algebra that allows for adaptation of filter and profile semantics. The implementation of this algebra as a profile definition language is presented here. A basic form of the adaptation of profiles as envisioned in Chapter 8 is implemented in A-MEDIAS. We abstain from the extended timestamping and event ordering method proposed in Chapter 6. Only a basic timestamping algorithm has been implemented, which may be extended later.

[1] The acronym A-MEDIAS is historically determined and contains a reference to our project named MediAS [Med03]. The system's name resolves to *A*daptable *medi*ating *A*lerting *S*ervice.

Section 9.4 presents the structure for the filter components in A-MEDIAS. The algorithms proposed in Chapter 7 are implemented in A-MEDIAS. We briefly describe their adaptability to different profile distributions as discussed in Chapter 8. Experimental efficiency tests for our filter implementation have been performed; their results are presented in Chapter 10.

9.1 A-MEDIAS Architecture

This section introduces the A-MEDIAS architecture. The A-MEDIAS system is a prototypical event notification system implemented in Java. The system realizes the event filtering methods introduced in this thesis and additionally contains components to load, store, and index profiles, to accept event messages, and to create and send notifications to clients.

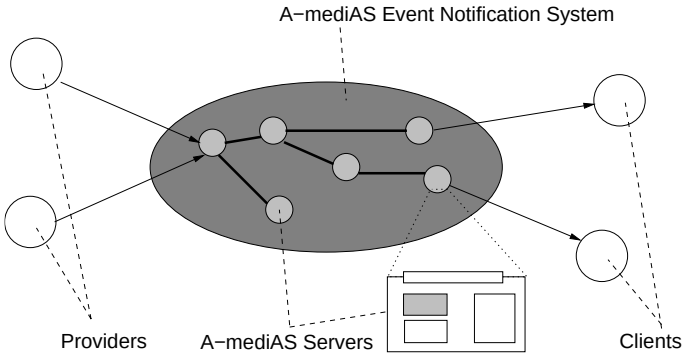


Figure 9.1: Architecture of the A-MEDIAS system

Figure 9.1 shows the architecture of the distributed A-MEDIAS event notification system. The system is formed by a number of servers that are connected by an acyclic overlay network. Each server implements an event notification system as discussed in this thesis. Additional components control the distribution of event and profile data over the network. In this thesis, we focus on the functionality within each ENS server. However, for the implementation of our distributed A-MEDIAS system, we make use of findings developed for distributed service: For the distribution of profiles and events, we implemented and tested three distribution methods. For a detailed analysis of the distribution aspect of A-MEDIAS see [Bit03].

The structure of each A-MEDIAS server, i.e., each node within the A-MEDIAS system, is depicted in Figure 9.2. The figure shows the components within the server and the data-flow between the components. A server node is contacted via a listener. Within a server node, each directly connected neighbor server within the network is represented by a *neighbor representation* object. The provider and client components are not shown here – they are handled similarly to neighbor objects. The neighbor object accepts event messages and profiles submitted from the respective neighbor node. The incoming profiles are redirected to the *distribution component*, which handles the profiles according to the selected distribution method. Based on these methods, the profiles are submitted to the *event notification component* or redirected to other A-MEDIAS servers (i.e., neighbors) in the network.

Incoming event messages are translated into event objects and redirected to the event notification

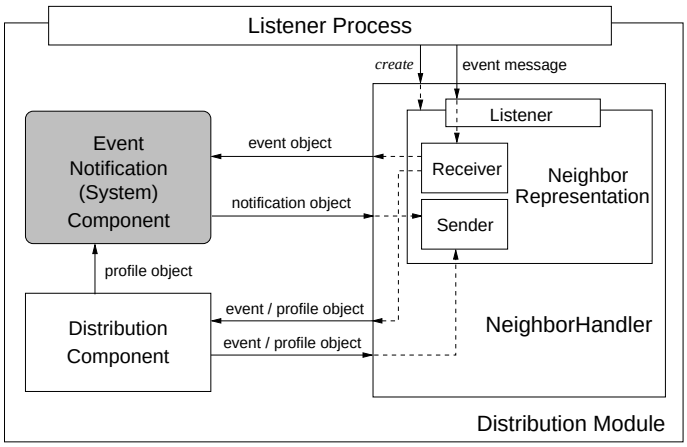


Figure 9.2: A-MEDIAS event notification server: Event notification component with distribution-related components

component. The event notification component observes and filters the events and sends notifications to the interested neighbors. Additionally, the distribution component may distribute event messages within the network. For detailed information about distribution methods see [Bit03].

The architecture of the event notification component is shown in Figure 9.3. The component can be used as a stand-alone centralized event notification service. Then, profiles and events are submitted as messages to the profile parser and event observer, respectively. If used as a component within a distributed system of A-MEDIAS servers, profiles and events are submitted as objects. Within the event notification component, the profile pool acts as filter engine.

Primitive profiles are inserted into the *primitive profile pool*: Each profile query in the profile tree references to the respective profiles. Composite profiles are inserted into the *composite profile pool*. The primitive profile parts of composite profiles are inserted into the primitive profile pool: either at once (when following a two-step filter method) or successively (when following our single-step method). The *profile references* of those contributing profiles are linked to the respective composite profile.

The filter engine follows the single-step method, the profile manager controls the alternative two-step filter method. Time events regarding composite events are managed in the *time profile pool*. The translation of client profiles according to application profiles is triggered by the evaluation of a composite profile. Application profiles are created by the application administrator based on source profiles.

Profiles regarding time events are inserted into the time profile pool. The pool is implemented as a heap, where the upmost element refers to the (temporally) next profile, i.e., to the next time event to be announced.

Qualitative adaptation is supported by client profiles, source profiles, and application profiles. The source profiles are not shown in the figure. Several *application profiles* may be defined, one of them is selected as the current application. Each application profile defines basic transformation rules for profile parameters for each of the different source profiles. Currently, adaptation of the event instance selection parameter EIS for each source profile is supported. Adaptation of the event instance consumption parameter EIC currently only supports a logging mode that results in notifications about all event pairs.

Incoming events are evaluated against the primitive profile pool. If the event matches a primitive

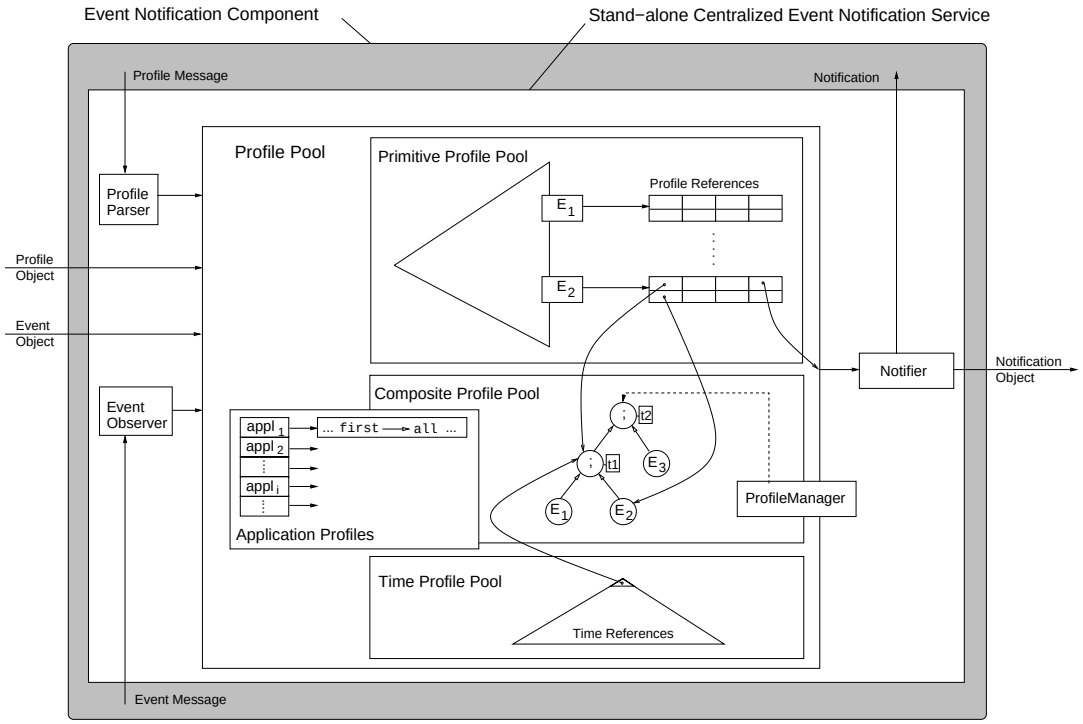


Figure 9.3: A-MEDIAS event notification component

profile, the notifier is triggered. If the event matches a contributing part of a composite profile, the next step in the filtering of the composite profile is triggered. A detailed description of the filtering of primitive and composite events is given in Section 9.4. The notifications are either sent to the neighbor component in the network component (for the distributed service) or directly to the client via email (for the centralized system).

9.2 Reference of Essential A-MEDIAS Classes

This section briefly describes the important classes and their interactions in form of an enumerative reference using UML (Unified Modelling Language) diagrams [Fow97]. Fragments of selected class definitions are given in Appendix A. Only the relevant attributes and methods are shown in the UML diagrams. Parameters in methods are omitted.

Providers and Source Profiles. Figure 9.4 shows the A-MEDIAS classes for event providers (Publisher) and their event sources. The sources are described by source profiles (Advertisement). The provider objects are controlled by a structure (PublisherPool) that references the provider objects. Publishers submit event messages that are transformed into event objects (Event). The events follow the event type referenced in the advertisement. The structure of a publisher object is shown in Code Fragment A.1 on Page 185. A section of the class definition for advertisement objects (implementing source profiles) is depicted in Code Fragment A.2. A section of the class definition for an event object is shown in Code Fragment A.3.

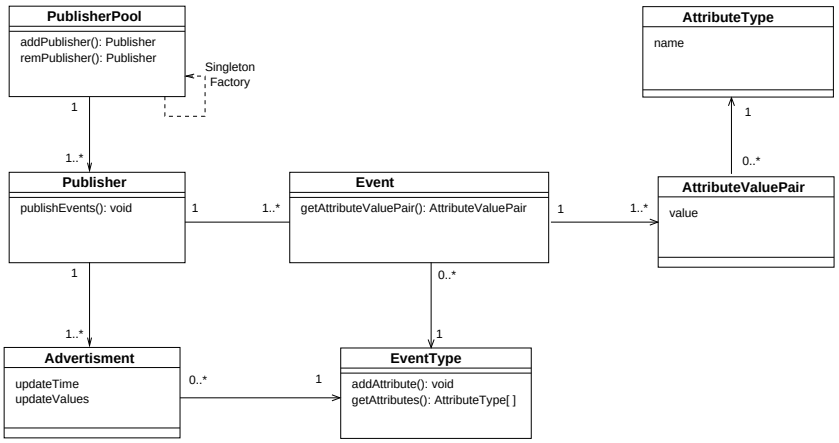


Figure 9.4: UML representation of providers (Publisher) in A-MEDIAS

Clients and Client Profiles. Figure 9.5 shows the A-MEDIAS classes for clients (Subscriber) and their profiles (Profile). The clients are controlled by a structure (SubscriberPool) that references the client objects. Clients may define one or more profiles. A section of the class definition for client objects is shown in Code Fragment A.4.

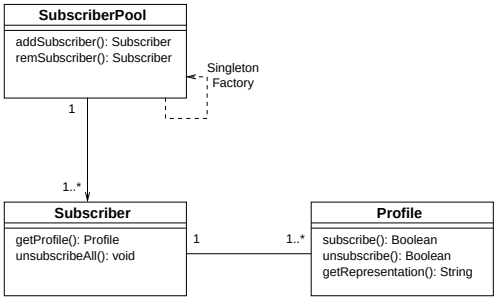
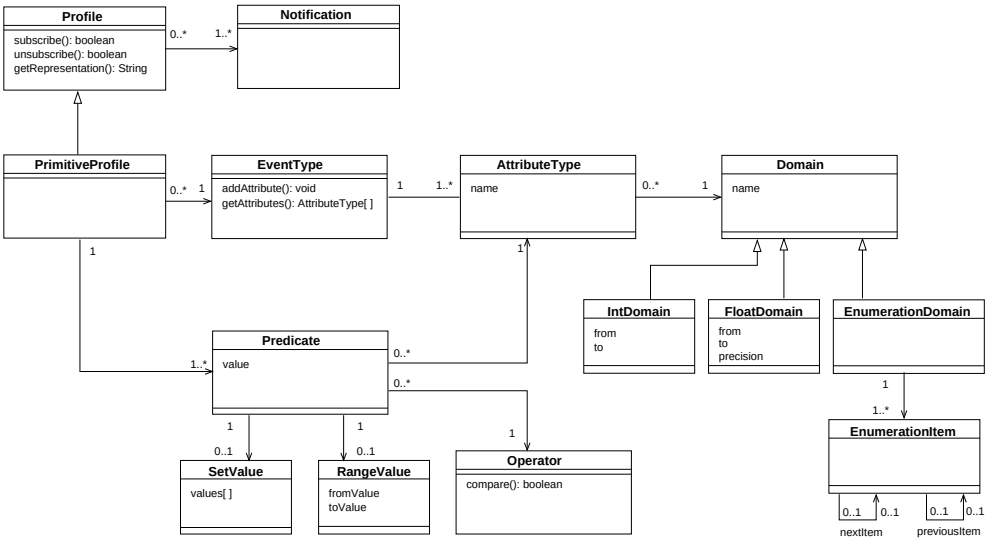
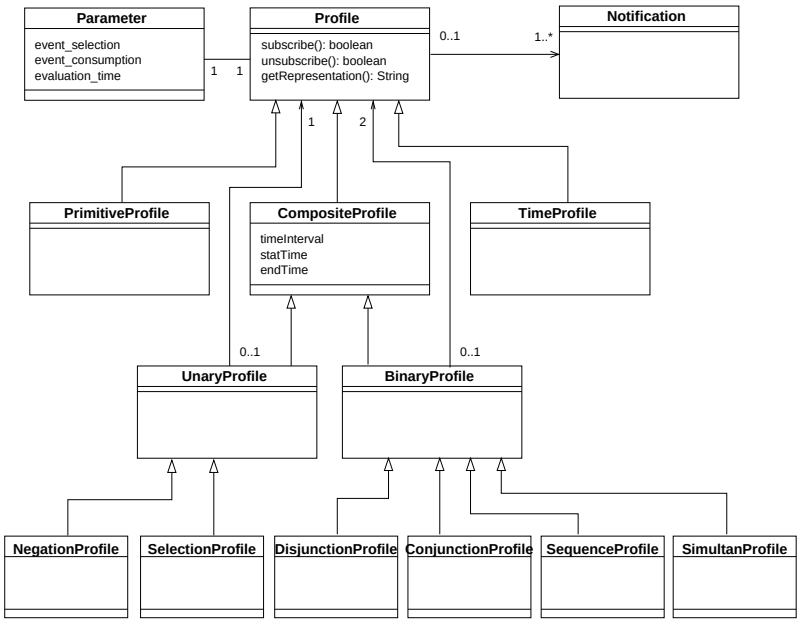


Figure 9.5: UML representation of clients (Subscriber) in A-MEDIAS

Primitive and Composite Profiles. A-MEDIAS distinguishes primitive and composite profiles. In A-MEDIAS, primitive profiles consist of predicates on attributes (see Figure 9.6(a)). Each profile may have several notifications, e.g., for different notification types (via email and via SMS). The structure of a profile follows the structure of the respective event message type (EventType). Currently, A-MEDIAS supports attribute value types (AttributeType) for integer (IntDomain), float numbers (FloatDomain), and an enumeration type (EnumerationDomain). A section of the class definition for primitive profile objects is shown in Code Fragment A.7. In A-MEDIAS, composite profiles may be unary or binary profiles referring to unary or binary event operators, respectively (see Figure 9.6(b)). A section of the class definition for composite profile objects is shown in Code Fragment A.8.



(a) Primitive Profiles



(b) Composite Profiles

Figure 9.6: UML representation of profiles in A-MEDIAS

Applications. Figure 9.7 shows the A-MEDIAS classes for applications and their transformations (TransformationSet). Each application holds a set of transformations. This transformation set depends on the available event sources: Transformations may be defined for each source profile. Source profiles are defined in advertisements (Advertisement). A section of the class definition for an application is shown in Code Fragment A.9.

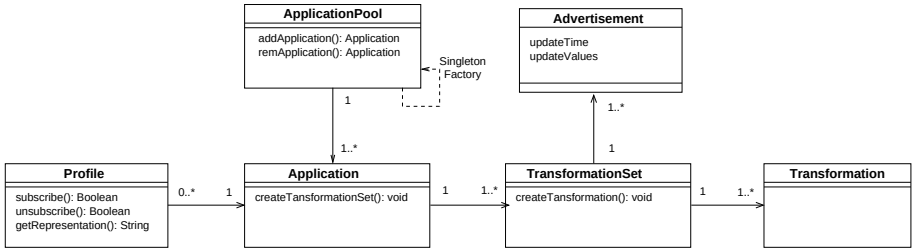


Figure 9.7: UML representation of applications in A-MEDIAS

9.3 A-MEDIAS Profile Definition Language

In this section, we introduce the grammar of the profile definition language for client profiles that is employed in the A-MEDIAS system. The language is shown in two parts: Figure 9.8 presents the main concepts, while Figure 9.9 gives auxiliary definitions.

For the specification of syntax and grammar, we use the Extended Backus-Naur Form (EBNF) proposed by Wirth [Wir77]. In EBNF, square brackets $[.]$ indicate zero or single occurrences of the enclosed expression and parentheses $(.)$ can be used to group expressions. The signs enclosed by quotation marks and the capitalized words are the terminals of the grammar. A `STRING` is a character sequence that may contain letters and digits. The structures of `REAL` and `INTEGER` follow the common rules for real and integer numbers, respectively. An `ENUMERATION` structure refers to strings or numbers with a predefined order. The language's grammar is a direct translation from our parameterized algebra (cf. Chapter 5, Page 63). As introduced there, the exact semantics of each composite operator is controlled by parameters. We included the additional parameter of event evaluation time in our profile definition language. This parameter has first been discussed in Chapter 5, Page 70.

A definition of a new operator for each different parameter setting would not be sufficient: The size of the profile language would increase significantly. Additionally, adapting the profiles would be a complex task due to the fixed semantics of these operators. Our approach follows the concept of polymorphism: The semantics of the basic operator is determined by the parameter setting, which may change during the system's runtime.

Profiles on primitive events are described using attribute-operator-value triples. The structure of profiles follows predefined event types that are advertised in source profiles. A *profile type* defines the attributes and their domains. The available attribute domains are numbers and enumerations of numbers or strings. The available operators support equality tests (=) and inequality tests (< and >). Additionally, the language supports the operator 'between' (`<> a and b`) and the set operator 'in' `->(a,b)` that refers to enumerations. Each predicate (`PrimExp`) refers to a single attribute. The attribute predicates are implicitly combined by conjunctions. For disjunctive predicates, a new profile has to be defined.

```
          Profile ::= 'PROFILE(' ProfContent ')'
      ProfContent ::= PrimProf | CompProf | SelectProf | NegProf
         PrimProf ::= 'PRI(' PrimExp [ ', TYPE=' EventType ] ')'
         CompProf ::= 'COMP(' ProfContent [ ',' EisParameter ]
                      CompOperator ProfContent [ ',' EisParameter ]
                      [ , MULTI =' EicParameter ] [ ',' TimeExp ]
                      [ , EVAL =' EetParameter ] ')'
          PrimExp ::= AttributeName AVtripleOperator Value
                      | AttributeName AVquadruple
          TimeExp ::= ' EVALTIME = ' TimeFrame
                      [ ', STARTTIME = TimeStamp ]
                      [ ', MAXTIME = TimeFrame ]
     CompOperator ::= 'OR' | 'AND' | 'BEFORE' | 'WITH'
       SelectProf ::= 'SELECT(' ProfContent ', NUMBER=' INTEGER
                      [ ',' EicParameter ] [ TimeExpresssion ] ')'
          NegProf ::= 'NOT(' ProfContent
                      [ ',' EicParameter ] TimeExp ')'
 AVtripleOperator ::= '=' | '>' | '<' | '>=' | '<='
      AVquadruple ::= '<>' Value 'and' Value
                      | '->' '(' Value ',' Value ')'
         TimeUnit ::= 'ms' | 's' | 'min' | 'h' | 'day' | 'week'
     EicParameter ::= 'All-PAIRS' | 'UNIQUE-PAIRS'
     EisParameter ::= 'FIRST' | 'LAST' | 'ALL' | INTEGER
     EetParameter ::= 'CONTINUOUS' | 'FINAL'
```

Figure 9.8: Profile Definition Grammar in A-MEDIAS. (Part 1/2)

The binary operators for composite events implement disjunction (`OR`), conjunction (`AND`), and sequence (`BEFORE`). Additionally, profiles for simultaneous events may be defined (`WITH`). Simultaneous events occur with a temporal difference below a certain epsilon. The unary operators implement selection (`SELECT(.)`) and negation (`NOT(.)`). Composite events may be nested.

The temporal restrictions of event compositions are defined as relative time references (`EVALTIME`). The evaluation of profiles starts either directly after the definition of a profile or at a `STARTTIME` defined within the profile. The profile is evaluated until the time span `MAXTIME` is elapsed. Time references are given in real time with reference to Greenwich Mean Time (see Figure 9.9). A timestamp referenced by `'0'` refers to the time of profile definition. The adaptive control of the accuracy in composite event detection and composition as proposed in chapters 6 and 8 has not been implemented in A-MEDIAS.

```
     EventType ::= STRING | 'TIMEPROFILE' [', TIMEZONE ='
                   ( '+' | '-' ) Hours '.'  Minutes ]
 AttributeName ::= STRING
         Value ::= AlphaNum | TimeStamp
      AlphaNum ::= ENUMERATION | REAL
     TimeStamp ::= (Hours ':'  Minutes ':'  Seconds)| '0'
     TimeFrame ::= '"' Integer timeUnit '"'
      TimeUnit ::= 'hours' | 'minutes' | 'seconds'
         Hours ::= 0 | 1 | ...  | 23
       Minutes ::= 0 | 1 | ...  | 59
       Seconds ::= 0 | 1 | ...  | 59
```

Figure 9.9: Profile Definition Grammar in A-MEDIAS. (Part 2/2)

9.4 A-MEDIAS Filter Engine

This section describes the filter engine of the A-MEDIAS system in detail. Figure 9.10 shows the inner structure of the primitive profile pool. For each available profile type, a separate typed profile tree is built. The dispatcher redirects the incoming events to the respective profile tree. Due to the working-memory demands of the original profile tree structure as proposed by Gough [GS95], we implemented a modified version of a profile tree. Our typed tree version is a combination of the original profile tree with the counting algorithm [FLPS00]: For each attribute, a single node is built (as opposed to the attribute level with multiple nodes in the original tree). The attribute values of the events are evaluated; the intersection of the n results refers to the matching profiles. For each attribute, an array refers to the values of the respective profile predicates.

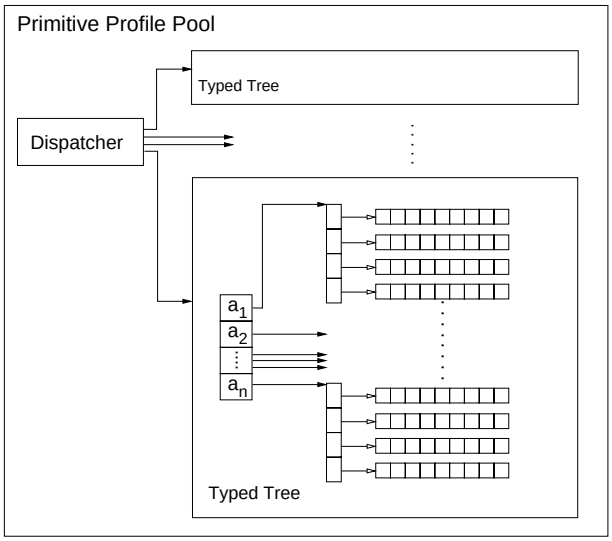


Figure 9.10: Primitive-event filters in A-MEDIAS

Each predicate value, in turn, refers to a list of profile references, i.e., to the profiles that contain the predicate. The profile reference list has been implemented in two version: as bit-list and as array. The array is used for a small number of profile references, whereas the bit-list is used for a higher number of references. The system changes automatically between the two reference systems if necessary.

For the reordering of the tree according to event and profile distributions, we implemented the selectivity measures A1 and A2: The ratio of the probability of the zero-subdomain and the probability of the domain-size under consideration of (A1) the profile distribution or (A2) the event and profile distribution (cf. Chapter 7, Page 105).

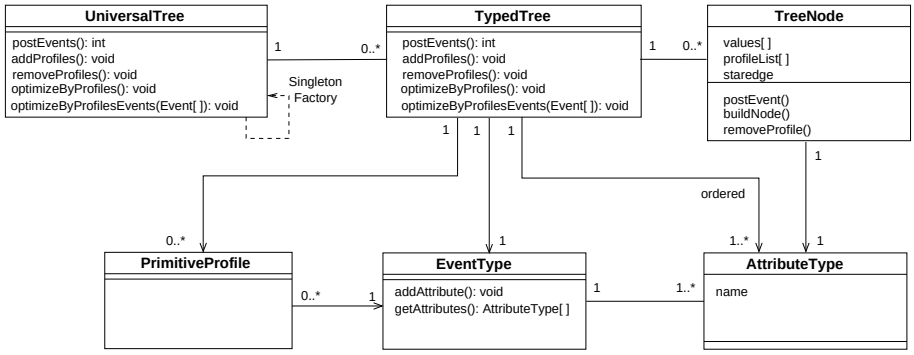


Figure 9.11: UML representation of the filter component for primitive events in A-MEDIAS

Figure 9.11 presents a simplified UML diagram for our A-MEDIAS event notification component. The dispatcher is implemented as a singleton (UniversalTree). The trees for each event type are objects of class TypedTree. Each attribute (AttributeType) in this tree is represented by a TreeNode. The reordering is a method of the universal tree and of the typed tree object (see Figure 9.11. The reordering of the tree according to measure A1 is implemented by method optimizeByProfiles, the reordering based on measure A2 (event and profile distribution) is implemented in method optimizeByProfilesEvents. Both optimizations consider the current profile set in the tree, optimizeByProfilesEvents additionally analyzes a given event set. The profile distribution is obtained based on the given profile set. The event distribution is derived from the representative set of events (parameter Event[ ]).

Quantitative adaptation is supported by optimization strategies for primitive-event and composite-event filtering. Currently, the tree-reordering has to be triggered by the system administrator. We plan to extend the current implementation to support auto-adaptation.

9.5 Summary

This section introduced our prototypical implementation A-MEDIAS of an adaptive integrating event notification service. We presented the general system architecture as well as details of the filter implementation. Additionally, we described the profile language of A-MEDIAS, which implements the event algebra introduced in Chapter 5. Results of efficiency tests for our algorithm implementations within a single A-MEDIAS server are presented in the next chapter. The performance of a network of A-MEDIAS servers has been analyzed in [Bit03].