Andreas M. Wahl, Thomas Fischer, Richard Lenz

MATINEE: A Quality-of-Service-aware Event Semantics Modeling Language

Technical Report CS-2013-07

2014

Please cite as:
Abstract—Distributed event-based systems have risen in significance over the last few years across many different application domains. Still, the configuration of available communication middleware solutions remains a tedious task driven by technical terms and manual performance optimization. We present the M^2 etis Quality-of-service-aware Semantics Modeling Language (MATINEE), a domain-specific language for modeling the Quality-of-Service (QoS) requirements and semantic properties of event-based systems in the terminology of the application domain. No knowledge of the technical terms of the underlying middleware solution is required by the application developers to conduct the configuration. Embedded in a novel configuration methodology provided by the Massive Multiuser Event InfraStructure (M^2 etis) project, MATINEE specifications are interpreted to automatically derive optimal middleware configurations without the need for manual optimization. The purpose of MATINEE is to provide a flexible language that can easily be adapted to different application domains. We focus on the language specification of MATINEE, briefly sketch our configuration methodology and present related language approaches.

Index Terms—domain-specific language, m^2 etis, quality-of-service, event semantic, publish-subscribe

I. INTRODUCTION

The optimal configuration of complex middleware solutions is a non-trivial task that requires profound technical knowledge of the inner workings of the middleware and the application domain it is deployed in. Distributed event-based systems utilize communication middleware solutions for event dissemination. Especially when QoS requirements are involved, determining an optimal configuration might turn into a tedious task of manual parameter tuning.

This introduces two challenges: On the one hand a language for the specification of middleware configurations is required that relies on as least middleware-specific terms and knowledge as possible. On the other hand an automation of the configuration workflow is desirable to minimize the impact of the middleware configuration on the development process of distributed applications.

In this paper, we focus on our proposed solution to the first challenge. We present the M^2 etis Quality-of-service-aware Semantics Modeling Language (MATINEE), a modeling language that allows developers to specify the configuration of event-based middleware on the level of application semantics. The language is adaptable to different domains with their diverse terminologies. It provides the expressiveness to model different event types in a domain-specific terminology. Moreover, the translation of semantic descriptions to the configuration of a certain middleware is definable in order to customize an automated deduction of the technical configuration.

In section II we illustrate how MATINEE is embedded into the automated configuration methodology of the M^2 etis project. Besides an overview of the artifact production workflow, we sketch the configuration basics. We give a brief overview of related approaches to QoS specification languages in Section III to motivate the development of MATINEE.

Based on this compact survey the MATINEE language specification is detailed in section IV. This specification covers the structure and constraints of the different language artifacts. It also explains how MATINEE artifacts can be interpreted during middleware configuration. Multiple artifact examples illustrate the productive use of the language. Therefore the specification is to be considered as a language reference that accompanies further publications in the context of the M^2 etis project.

For the exemplification of the different artifacts of MATINEE we employ the application domain of Massively Multiplayer Online Games (MMOGs). These games enable the interaction of geographically distributed players in a virtual world. For a detailed introduction of this application domain we refer to our previous work where different use case scenarios and common requirements are discussed [1].
II. M²ETIS CONFIGURATION METHODOLOGY

MATINEE is an integral part of a configuration methodology for a quality-of-service-aware design-time configurable publish-subscribe middleware provided by the M²etis project. The basic idea behind this methodology has been discussed in our previous work [2]. A reference implementation is publicly available\(^1\).

The configuration methodology aims for extensive configurability without significant performance impact compared to custom implementations. To achieve this, configurability is limited to the design-time. The configuration effort is minimized while configuration remains intuitive for application developers. The description of the middleware configuration is performed in the terminology of the application domain.

On a technical level the behavior of the middleware is configured by strategies that allow to encapsulate certain aspects of event dissemination. Strategies may concern aspects such as routing, event filtering or synchronization. A configuration is a composition and parameterization of such strategies.

Our configuration methodology requires several different types of semantic artifacts that are created as the middleware and application development progresses. An overview of the artifact production workflow is depicted in Figure 1. The workflow is separated by the different roles developers can assume and the MATINEE artifacts they produce.

During middleware development properties and capabilities of the different strategies are denoted in corresponding strategy descriptions (cf. Section IV-B). This task is performed by middleware developers, who have insight into the inner workings of the middleware and the implementation of each strategy.

To use the middleware in a domain-specific context, domain experts describe the specific semantics and the terminology of a certain domain. Two distinct artifacts are produced: the domain profile containing the semantics and terminology of the domain (cf. Section IV-C) and the network profile describing the assumed network characteristics (cf. Section IV-D).

Finally, using the domain and network profile, application developers can describe different event types to be disseminated using the middleware, in their familiar domain-specific terminology. The resulting artifacts are called event type descriptions (cf. Section IV-E).

All of the introduced artifacts form the semantic description of a distributed application. The information gathered in the different artifacts is used to enable an automated configuration process.

\(^1\)The source code of the reference implementation is available at https://code.google.com/p/m2etis/.

As depicted in Figure 2, the semantic description is interpreted by a central configuration component, called \textit{M²etis Adaptive System Configurator} (MAESTRO). The purpose of MAESTRO is the generation of a cus-
tom M2etis middleware library. MAESTRO employs discrete-event simulations and machine learning concepts to deduce optimal compositions and parameters that fulfill the requirements introduced by the semantic description. To achieve this goal, MAESTRO uses pre-computed meta-models of the application domain constructed from simulation results to generate the configuration of the M2etis middleware. Domain and network profiles are used to parameterize the required simulation scenarios to be conducted by the simulator component. Wahl [3] provides further details about the interpretation of the different MATINEE artifacts and the automated configuration workflow.

III. RELATED LANGUAGE APPROACHES

An important prerequisite for the adaptability of a middleware is an efficient method to specify QoS requirements for the network communication of an application. Ensuing from the coarse overview of the configuration methodology which MATINEE is embedded in, we present a compact survey on different existing approaches for application-driven specification of QoS requirements for communication middleware solutions. Languages designed for this purpose offer modeling capabilities on different abstraction layers. The use of different language paradigms enables a wide variety of individual solution concepts.

The discussion of related approaches is limited to languages that are especially suited for the specification of requirements on the level of applications. We omit depicting exemplary specifications for clarity reasons. For detailed examples of the different approaches and their productive usage we refer to the original publications. Additional taxonomies and comparisons of different QoS languages can be found in the work of Jin und Nahrstedt [4] or Kritikos et al. [5].

A. Hierarchical QoS Markup Language

The Hierarchical QoS Markup Language (HQML) is a language for the specification of QoS requirements of internet-based multimedia applications [6]. HQML documents can be exchanged between communication parties in order to negotiate specific QoS parameters at runtime. The language definition is based on the Extensible Markup Language (XML). This enables an easy extension of the language by adding further QoS parameters.

Documents written in HQML consist of hierarchically nested tags that characterize the structure of the modeled system as well as the requirements of single system components. Attributes are used to annotate tags with relevant metadata.

The requirement specification can be performed on different abstraction levels. For this purpose HQML provides schemas for QoS requirements from an end-user perspective, for requirements on application level and for requirements on the level of network resources.

The composition of a specification can be simplified by using the custom tool QosTalk [7]. QosTalk is capable of verifying the consistency of the specification.

B. QoS Modeling Language

With the help of the QoS Modeling Language (QML), QoS requirements of distributed applications can be described using object-oriented concepts [8]. The language aims for deployment in the context of object-oriented communication middleware. The focus of this approach lies in the static specification of QoS requirements and not in the description of the behavior at runtime in case of a violation of the QoS requirements.

QML is strictly typed and offers inheritance which fosters the reusability of QoS specifications. Child specifications may extend or override inherited requirements.

A QoS specification, written in QML, consists of three different artifacts: A contract type describes dimensions which can be used to classify the considered QoS aspects. Instances of a type are called contract and fill the the schema defined by the type with specific parameters. Profiles are used to connect contracts and the interfaces and operations of the corresponding distributed application.

C. Attribute Framework For provIding conteXt

The Attribute Framework For provIding conteXt (AFFIX) is used by the Family of Adaptive Middleware for autonomOUs Sentient Objects (FAMOUSO) project to specify QoS requirements and filter predicates [9]. In the following we focus on the possibilities to define QoS requirements.

AFFIX distinguishes between two different types of QoS specifications: First, the capabilities of the used network can be characterized. The maximum value for each network parameter is employed to describe the performance in edge cases. Second, it is possible to describe the application itself. For the application-level specification the QoS parameters are defined by minimal values that have to be maintained at all time to ensure the operation of the distributed application.

On a technical level, AFFIX employs C++ class templates in order to express QoS specifications. QoS
properties are represented by template parameters. The specification is performed directly in the source code of the distributed application.

Additionally, AFFIX offers the possibility to statically validate the satisfiability of the requirements. For this purpose, the network-level maximum values are compared to the application-level minimum values of the different QoS parameters. As this comparison is performed using template meta programming, invalid specifications can be already discovered during the compilation process.

D. Distributed QoS Modeling Language

The Distributed QoS Modeling Language (DQML) [10], [11] is developed in the context of the ADaptive Middleware And Network Transports (ADAMANT) project [12], [13]. The language offers mechanisms for the description of QoS properties of entities participating in distributed object-oriented architectures.

DQML supports application developers during the creation of the entities relevant for the QoS specification. Such entities can, for example, be publishers, subscribers, or additional information sources and sinks. These entities can then be connected to QoS specifications.

The specifications consist of a combination of properties that are selected from a collection of predefined templates. By an individual parameterization each property can be tailored to the requirements of a certain application.

The creation of the specification is supported by the Generic Modeling Environment [14] which provides a graphical user interface. This tool also validates the QoS requirements regarding to completeness, consistency and applicability.

E. Quality of Service Architecture

The Quality of Service Architecture (QoS-A) allows for the specification of QoS requirements in form of C-style structures [15]. The required information is modeled using scalar parameters which are composed to a QoS contract.

QoS-A offers the possibility to describe limits for network metrics, the qualitative characterization of QoS requirements and the prerequisites for the adaptation and the preservation of the defined requirements. Additionally, reservations of network resources may be modeled.

F. Fuzzy Control Model

Li and Nahrstedt follow a rule-based approach for the specification of QoS requirements [16]. The resulting Fuzzy Control Model employs runtime adaptation by a fuzzyfied configuration component that evaluates QoS specifications.

Information on the requirements is not directly defined by static parameters, it is rather specified indirectly with the help of rules. The first part of a rule describes those conditions that have to be fulfilled before an adaptation is necessary. All system parameters which are measurable at runtime are accessible from this part.

The second part of a rule describes the reaction of the system if the required conditions are met. To ensure adaptability, actions on the application and middleware level can be performed or QoS parameters can be modified.

G. Discussion

The examined description languages with their different language paradigms provide interesting approaches for the specification of QoS requirements. However, most of them favor the dynamic negotiation of QoS contracts at runtime which contrasts the design-time methodology of the M²etis project.

AFFIX, DQML, HQML und QoS-A are based on existing languages or development environments and therefore support an intuitive usage and efficient extension of the language. HQML, QML und DQML offer possibilities to describe QoS requirements on a high abstraction level but still require the developer to manually map the application context to QoS requirements.

Existing languages lack the full capabilities to express a domain-specific context and the corresponding mapping onto QoS requirements and technical parameters required by our automated configuration methodology. As a consequence a custom modeling language is necessary to be used in the M²etis project.

IV. LANGUAGE SPECIFICATION

MATINEE is a domain-specific schema for the data serialization language YAML ain’t Markup Language (YAML)². To model the previously introduced semantic artifacts, YAML is extended with custom data types and individual content structures. Each MATINEE artifact consists of hierarchically nested mappings (associative arrays) and lists of mappings. The basic structural evaluation of the artifacts can be done by a standard YAML processor while the the middleware configuration component MAESTRO is responsible for the evaluation of the actual artifact content.

²The formal YAML language specification is publicly available at http://www.yaml.org.
The following sections provide detailed specifications of the different artifacts of MATINEE. The structure of each artifact is textually described and graphically illustrated. All constraints that apply to the artifacts are also detailed.

To ease the comprehensibility of the artifact structure we present a sample instance of each artifact and explain its semantics. We employ examples from the application domain of MMOGs and the reference implementation of the M²etis communication middleware.

**Notation Remarks**

To describe the structure of the different MATINEE artifacts in a generic way a method to specify schemas for YAML is required. As there is currently no standardized schema language for YAML available, we rely on UML class diagrams to illustrate the artifact schemas. This is feasible as YAML documents can be considered as serialized object representations that do not have active behavior. A similar visualization approach is used by Cánovas and Cabot [17] to represent the structure of documents written in the Java Script Object Notation (JSON) which is a subset of YAML.

Each MATINEE artifact is depicted as a UML class. The class name corresponds to the artifact type. Artifact contents are modeled as class attributes. If the value of an attribute is of a complex custom data type, this type is also described using a separate UML class which is named according to the referencing class attribute. Otherwise an existing data type is assigned to the attribute. If the schema and identifier of a complex data type can be dynamically defined during artifact specification the data type is represented as an <<abstract>> class. If a complex data type does not have a unique parent identifier it is depicted as an <<anonymous>> class.

Value constraints for artifact attributes are specified textually. Where applicable, all conditions that must be true, if the artifact is to be considered valid are explained.

We introduce a notional global class Middleware as depicted in Figure 3 that encapsulates the technical knowledge which is gathered by MAESTRO during the automated configuration workflow. This allows for concise and comprehensible constraint explanations.

The attribute Middleware.StrategyTypes contains all strategy types that are supported by the communication middleware. For the configuration workflow only strategies that belong to one of the types in this list are considered. The strategy descriptions of all those strategies that are available during configuration are accumulated in a list under the attribute Middleware.Strategies. Values for a wide range of parameters are gathered in different steps of the configuration workflow. These parameters and their respective values are collected in form of mappings using the attribute Middleware.SystemParameters. Middleware.Application and Middleware.Network represent the domain profile and network profile to be used as foundation for middleware configuration.

**A. Artifact Metadata**

Each MATINEE artifact instance is annotated with relevant metadata. These annotations enable the identification of different artifact instances and enhance the comprehensibility for human readers. Figure 4 illustrates the structure of these Metadata annotations.

Each metadata annotation must contain at least an Identifier to provide a unique name for the respective instance. It can be used for differentiation purposes when evaluating the MATINEE artifacts during middleware configuration. The uniqueness of the identifier must be guaranteed by the developer within the set of all artifacts of a certain type.

All other attributes are optional and may be left unspecified. Explanatory textual comments describing the purpose of the respective artifact can be provided using the attribute Description. Author designates a human stakeholder responsible for the artifact instance. Date and Version can be used to record the development progress of the artifact.
Example
Listing 1 shows an example of a completely specified metadata annotation for an event type of a MMOG. According to the provided information the event type is identified by the unique name Movement. The descriptions states that events of this type are used to distribute information about player movement. The artifact has been lastly modified on August 15, 2013 by the named author and is currently in Version 1.2.

```
Metadata:
  Identifier: Movement
  Description: Player movement in the virtual environment.
  Author: Andreas M. Wahl
  Date: 08-15-2013
  Version: 1.2
```

Listing 1. Metadata annotation for Movement

B. Strategy Descriptions

Each strategy implementation that is supposed to be used during middleware configuration must be accompanied by a strategy description. The generic structure of these descriptions is depicted in Figure 5.

```
Strategy
  + Metadata: Metadata
  + Classification: List<Mapping>
  + Compatibility: Compatibility
  + Configuration: Configuration
  + Exclusions: List<String>
```

```
Configuration
  + Information Class: String
  + Parameters: List<String>
```

```
Compatibility
  + Requirements: List<String>
  + Exclusions: List<String>
```

Fig. 5. Structure of Strategy descriptions

The specification of relevant metadata is mandatory. The artifact identifier must be unique among all strategy descriptions. Additionally, information concerning the classification, compatibility and configuration of a strategy implementation is to be modeled.

1) Classification: Each strategy must be annotated with classification attributes to be considered during the configuration process using the Classification attribute. As the middleware provides different types of strategies, which realize different aspects of the event dissemination, a mapping with the key Type is mandatory to enable middleware composition during configuration. The corresponding value must be one of the types supported by the middleware that are included in Middleware.StrategyTypes.

The strategy classification can be extended by additional parameters. These mappings allow for a more detailed characterization among all strategies of a specific type. The keys and values of these mappings may be arbitrarily chosen by the middleware developers.

2) Compatibility: Strategies are combined with each other to form a middleware configuration. To ensure a valid configuration, dependencies and incompatibilities to other strategies can be considered by specifying the optional attribute Compatibility.

If a strategy always requires strategies of specific types as combination partners these types are specified as an optional Requirements list of strategy types. Each list entry must be one of the types supported by the middleware from Middleware.StrategyTypes. Incompatibilities with specific strategy implementations are optionally specified in an Exclusion list of strategy identifiers. Each list entry refers to the identifier of a strategy implementation of the middleware as specified in the corresponding strategy description. A strategy from Middleware.Strategies is considered incompatible if its identifier matches one of the list entries.

3) Configuration: The technical information about the usage of a specific strategy implementation is encapsulated in a separate Configuration attribute.

Information that is required at run-time can be accumulated in a separate source code file. This optional Information Class can be specified to make the information available for configuration. The value of this attribute must be the name of a valid C++ class definition located in the scope of the middleware source code.

Some strategies support explicit configuration parameters to modify their run-time behavior. Values for these positional parameters can be supplied using the optional Parameters list. Allowed values are scalars and mathematical expressions that can be evaluated by incorporating the contents of Middleware.SystemParameters.

Example
To illustrate the use of strategy descriptions, we show an exemplary description of the strategy MTPOrder from the prototype implementation of M2etis in Listing 2. The strategy is assigned to the strategy type Order. No further attributes are necessary to classify the strategy.

MTP [18] requires that as few messages as possible get lost during transport in order to maintain an order among the disseminated events. To ensure this, we state as Requirements that MTP must be only used in
conjunction with a Delivery strategy which is responsible for resending lost messages. MTP is not incompatible with any other strategy, the list of Exclusions is therefore empty.

Additional information is required for the technical configuration of MTP. The C++ class MTPOrderInfo contains source code which is mandatory to use the strategy at runtime. The two additional configuration parameters specify a timeout in microseconds after which a message is considered as lost and the behavior if a message arrives too late.

--- !Strategy

Metadata:
  Identifier: MTPOrder
  Description: Protocol to guarantee atomic multicasts. See RFC 1301 for details.
  Author: Andreas M. Wahl
  Date: 05-11-2013
  Version: 0.9

Classification:
  - Type: Order

Compatibility:
  Requirements:
  - Delivery
  Exclusions: []

Configuration:
  Information Class: "m2etis::message::MTPOrderInfo"
  Parameters:
    - 1000000
    - "m2etis::pubsub::order::LateDeliver::DROP"

Listing 2. Strategy description for MTP

C. Domain Profiles

Domain profiles contain all information related to the semantics of a specific application domain that are necessary to conduct the middleware configuration. Figure 6 illustrates the extensive content of these artifacts.

Besides the mandatory specification of relevant metadata, all domain-specific parameters that should be available to derive a configuration decision must be defined in the domain profile. Furthermore the dimensions of the multidimensional classification schema for event types are specified in detail to be used by the application developers. To define the limits of performance metrics suitable for a specific application domain, common QoS requirements are modeled.

1) Domain Parameters and Parameter Presets: Information used to derive a configuration decision is represented in form of parameters. The ParameterDefinitions are part of the domain profile. MATINEE provides support for Scalars as well as parameter Presets.

   a) Scalar Parameters: All parameters that are required to derive simulation scenarios of the application domain during the middleware configuration must be specified. To determine which values ranges the requirements of an application domain are located, limits for these parameters are mandatory.

   Additional parameters can be optionally defined that allow to model the requirements of event types in the terminology of the application domain. These parameters can also be used to determine parameters required for the simulations based on value transformations. Hereby the technical information required to conduct simulations can be provided without leaving the domain terminology.

   MATINEE supports both parameters that are based on standard data types and parameters with explicitly enumerated codomains. If a parameter is based on a standard data type it is mandatory to specify the underlying Type. Limits for Minimum and Maximum values, as well as the Step distance between values can be specified optionally. A meaningful Default value may be specified to be used if the user does not provide a custom value. By supplying a list of expressions that can be evaluated using the contents of Middleware.SystemParameters, the Deduction of the parameter can be controlled. These expressions must be evaluable to a valid parameter value at run-time. They can be used to dynamically deduct a parameter value from other parameter assignments. Arbitrary mathematical operations are allowed for these expressions.

   If the codomain of a parameter is to be explicitly specified only the Values and the Type are mandatory. Default values and Deduction expressions are optional while no other attributes are permitted.

   b) Parameter Presets: To allow for an intuitive use of frequently required parameter assignments, the domain profile supports parameter Presets. A Preset combines assignments to multiple parameters under a unique identifier which can be referenced from event type descriptions. Each parameter assignment is represented as a mapping, where the key must be the identifier of a parameter specified under ParameterDefinitions.Scalars and the value must be within the codomain of the respective parameter.

   2) Dimension Definitions: Besides the definition of relevant parameters, the specification of the classification Dimensions to be used for configuration is an integral part of domain profiles. MATINEE offers a range of
model elements to modify existing dimensions and to create new ones for specific application domains.

A classification dimension is characterized by a set of dimension-specific parameters. Dimensions are used to select appropriate strategies that adhere to the requirements of event types during the automated configuration workflow. Rules can be defined for each dimension which trigger the selection of certain strategies if the dimension parameters specified by the event type fulfill certain conditions. To allow for an intuitive classification by the application developers, classes can be defined for each dimension.

a) Dimension Parameters: MATINEE offers multiple options for the definition of Dimension Parameters to provide developers with flexible mechanisms to create custom classification dimensions. Similar to the parameter definitions introduced before, parameters based on standard data types and parameters with explicitly enumerated codomains are permitted. The same constraints as defined above also apply to dimension parameters.

Dimension Parameters can be used to derive additional information during configuration. The values of dimension parameters can be mapped to value assignments of other parameters. This requires the specification of Transformation Rules. Each transformation rule consists of two parts: The precondition When states under which circumstances a transformation rule is applied, while the postcondition Transform contains the parameter assignments to be conducted.

Preconditions must be expressions that can be evaluated to a boolean value. They may contain operations that are defined on the codomains of the involved parameters. Each parameter assignment of a precondition is represented as a mapping. The mapping key must be the identifier of a value of a Dimension Parameter. The values to be assigned can be constants or mathematical expressions that can be evaluated to a valid parameter value. Each expression must be evaluable using the contents of Middleware.SystemParameters.

b) Strategy Selection Rules: Similar to transformation rules, additional rules can be specified to control the selection of relevant strategies during configuration. Each Strategy Selection Rule consists of a precondition and a postcondition. The precondition When describes under which circumstances the rule is triggered. It must be an expression which can be evaluated to a boolean value. Only operations on the Dimension Parameters are allowed. The postcondition Select defines the conditions which must be fulfilled by strategies to be considered during configuration. Only operations on the classification attributes of strategies from

Fig. 6. Structure of Domain profiles
Middleware.Strategies are permitted that return a boolean value.

c) Dimension Classes: Classes of a specific dimension can be modeled as value assignments to the corresponding dimension parameters. The assignments of each Dimension Class are contained in a uniquely named list of mappings. Each key must contain the identifier of a Dimension Parameter and each value must be a valid assignment to the parameter.

3) QoS Requirements: The basic QoS Requirements of an application can be described in form of allowed codomains of QoS metrics. This codomains can be considered during the automated configuration workflow and influence the decision process.

For every relevant QoS metric, a list entry is added to the attribute QoS. Every entry must contain an assignment to the mandatory attributes Minimum and Maximum.

Example
For a better understanding of the extensive modeling options for the dimensions of the multidimensional classification schema, we introduce three exemplary Dimension instances. Hereby, we use a generic classification schema, which is suitable for MMOGs. The purpose of this schema is to demonstrate the expressiveness of the modeling capabilities provided by MATINEE.

a) Context: The first exemplary dimension is the Context dimension. It is defined as shown in Listing 3. The dimension is characterized by three parameters.

Scale describes the number of nodes participation in the communication. Three different parameter values are permitted: small, medium or large. The default is medium. These abstract values can be mapped to numeric parameters using transformation rules. The corresponding numerical values can be derived using an expression depending on the available upstream data rate and the size of the transmitted payload. During configuration the values for these parameters can be dynamically collected from Middleware.Network and the respective event type description (cf. Sections IV-D and IV-E).

The parameter Velocity defines the number of events that are published per second and publisher. Possible values in this example are low, medium or high. Again, medium is defined as the default value. These abstract values are statically mapped to numerical values that are common for MMOGs. In this case, they are mapped to a range between 1 and 30 events per second and publisher.

Parameters:
- Velocity:
  Values: [low, medium, high]
  Default: medium
  Transformations:
  - When: Velocity == "low"
    Transform:
    - Velocity: 1
  - When: Velocity == "medium"
    Transform:
    - Velocity: 15
  - When: Velocity == "high"
    Transform:
    - Velocity: 30
- Scale:
  Values: [small, medium, large]
  Default: medium
  Transformations:
  - When: Scale == "small"
    Transform:
    - Scale: ceil(0.05*(Upstream/(Velocity*Payload)))
  - When: Scale == "medium"
    Transform:
    - Scale: ceil(0.1*(Upstream/(Velocity*Payload)))
  - When: Scale == "large"
    Transform:
    - Scale: ceil(0.3*(Upstream/(Velocity*Payload)))
- Cardinality:
  Values: [one-to-many, few-to-many, many-to-many]
  Default: many-to-many
  Transformations:
  - When: Cardinality == "one-to-many"
    Transform:
    - PublisherSubscribers: 1

The third parameter Cardinality describes the composition of a group of nodes characterized by the ratio between publishers and subscribers. The value one-to-many describes the situation that only one node acts as a publisher and all other nodes are only subscribers. Few-to-many covers the case where a few nodes publish and the remaining nodes only subscribe. Finally, many-to-many means that all nodes act as publishers as well as subscribers. The depicted transformation rules reflect these semantics by using a weight factor to determine the group composition.

Only one strategy selection rule is required for this dimension. It expresses that all strategies that are associated with the type Routing must be always considered for configuration decisions. This rule is necessary, because without a Routing strategy it is not possible to disseminate any events.

The three dimension parameters are used to define two exemplary dimension classes: The class Large-scale movement-like corresponds to the requirements of an event type for exchanging positions in the virtual world could have in an MMOG. The class Global announcement reflects a use case where a server distributes coordination events to all local players.
- Subscribers: Scale
  - PublisherSubscribers
  - When: Cardinality == "few-to-many"
  - Transform:
    - PublisherSubscribers: ceil(0.3* Scale)
    - Subscribers: Scale
  - PublisherSubscribers
- Subscribers: Scale
  - When: Cardinality == "many-to-many"
  - Transform:
    - PublisherSubscribers: Scale
    - Subscribers: 0

Strategy Selection:
- When: True
  Select: Type == "Routing"

Classes:
- Large-scale movement-like:
  - Scale: large
  - Velocity: high
  - Cardinality: many-to-many
- Global announcement:
  - Scale: large
  - Velocity: low
  - Cardinality: one-to-many

Listing 3. Definition of the Context dimension

b) Synchronization: Another example for a relevant dimension in the context of MMOGs is the synchronization dimension. A MATINEE excerpt is shown in Listing 4. The consistency requirements of the application can be modeled using the three boolean parameters Order, Delivery and Compensation.

Three strategy selection rules are sufficient to model the configuration semantics of the dimension. If Order has the value True, all strategies with the type Order are considered for configuration. If Delivery has the value True, Delivery strategies are considered for configuration to ensure that no events are lost during dissemination. The Compensation parameter expresses that strategies are selected which have the type Delivery and are annotated with a parameter Compensation with the value True. Hence, only a subset of delivery strategies is considered, namely those that allow for compensation of invalid events.

Two possible classes give an impression of the semantics of this dimension. Of course it is also possible to specify more permutations of dimension parameters as separate classes if required by the distributed application.

Parameters:
- Order:
  - Type: bool
  - Default: False
- Delivery:
  - Type: bool
  - Default: False
- Compensation:
  - Type: bool
  - Default: False

Strategy Selection:
- When: Order == "True"
  Select: Type == "Order"
- When: Delivery == "True"
  Select: Type == "Delivery"
- When: Compensation == "True"
  Select: Type == "Delivery" and Compensation == "True"

Classes:
- Wait, throw away:
  - Order: True
  - Delivery: False
  - Compensation: False
- None:
  - Order: False.
  - Delivery: False
  - Compensation: False

Listing 4. Definition of the Synchronization dimension

c) Validity: Two parameters are sufficient to define the semantics of the validity dimension in MATINEE (cf. Listing 5). Progress defines in which unit the validity of an event is measured. The value Time specifies a time-based validity, while Event defines validity as a discrete counting of events. None is the neutral element and is used for the explicit deactivation of the strategy type Validity.

The validity interval of an event is defined by the parameter Duration. All possible values are defined either in microseconds or as a number of events, depending on the value of Progress.

The transformation rules are used to map the abstract values to numeric constants. The strategy selection rules also consider the value of Progress and accordingly select the appropriate subset of strategies.

Again, we define some self-explanatory classes for this dimension. In this example Short-lived timer and None are shown.

Parameters:
- Progress:
  - Values: [none, time, event]
- Duration:
  - Values: [none, short, medium, long]
  - Default: medium

Transformations:
- When: Progress == "time" and Duration == "short"
  Transform:
  - Duration: 50000
- When: Progress == "time" and Duration == "medium"
  Transform:
  - Duration: 100000
- When: Progress == "time" and Duration == "long"
  Transform:
  - Duration: 300000
- When: Progress == "event" and Duration == "short"
  Transform:
  - Duration: 2
- When: Progress == "event" and Duration == "medium"
Transform:
- Duration: 5
- When: Progress == "event" and Duration == "long"
Transform:
- Duration: 10

Strategy Selection:
- When: Progress == "time"
  Select: Type == "Validity" and Progress == "Time"
- When: Progress == "event"
  Select: Type == "Validity" and Progress == "Event"

Classes:
- Short-lived timer:
  - Progress: time
  - Duration: short
- None:
  - Progress: none
  - Duration: none

Listing 5. Definition of the Validity dimension

Example
Listing 6 shows an exemplary domain profile for our use case scenario of MMOGs that is conform to the MATINEE language specification. All defined parameters are either relevant for the specification of dimensions or form the domain-specific terminology of MMOGs.

The parameters PublisherSubscribers, Velocity and Payload are technically motivated and relevant for the execution of simulations. PlayerDensity and VisualRange are examples for domain-specific parameters that may be used to describe event types intuitively. The definitions of the three different classification dimensions have been discussed before and are omitted for conciseness.

The specified QoS requirements state that for an application in the domain of MMOGs the average latency may not exceed 300 milliseconds. Additionally, the average event loss may not exceed 50 percent.

Listing 6. Domain profile for MMOGs

D. Network Profiles

Distributed applications rely on assumptions about the underlying network used for communication between the application nodes. MATINEE Network profiles provide an extensible schema to capture network-related information as illustrated in Figure 7.

Fig. 7. Structure of Network profiles

Network profiles consists of two parts: The metadata annotation is obligatory and must incorporate an identifier that is unique among all network profile instances. The specification of network parameters is optional. Network parameters that are known or can be estimated at design-time are gathered in a list of mappings.

The key of each mapping must be an identifier of a network parameter that is supported by the middleware implementation. The corresponding value must be within the parameter codomain as defined by the middleware implementation. If required parameters are missing or no
parameters are specified at all, it lies in the responsibility of the configuration tool that evaluates the artifact to determine meaningful default values.

**Example**

Listing 7 shows an exemplary network profile. All nodes in the network connect via a medium-grade broadband internet connection. The latency between the equally-distributed nodes is estimated by 30 ms. For each node a downstream data rate of 6 Mbps and an upstream data rate of 1 Mbps is available. No further assumptions about message loss or network jitter are made.

--- !Network

Metadata:
Identifier: BroadbandNetwork
Description: This network consists of equally-distributed nodes connected via broadband.
Author: Andreas M. Wahl
Date: 08-15-2013
Version: 1.0

Parameters:
- Hop-to-Hop Latency: 30ms
- Downstream: 6Mbps
- Upstream: 1Mbps

Listing 7. Network profile for BroadbandNetwork

**E. Event Type Descriptions**

To customize the middleware according to the requirements of the event types of the distributed application, developers must describe these in a structured way. MATINEE provides a flexible schema for **Event Type Descriptions** (cf. Figure 8).

As for all artifacts, the specification of metadata is mandatory. Furthermore the payload structure of the events to be disseminated, the multidimensional classification of the event type, additional value assignments to parameters and QoS requirements can be described.

1) **Payload:** The optionally specified **Field** instances for the **Payload** of the event type can be considered when selecting a QoS-optimal configuration. During middleware configuration corresponding source code is generated to make the payload structure available to the distributed application.

Two different types of **Fields** are supported: range fields and value fields. Range fields must be based on a standard data type and can restrict the domain of possible field values by specifying allowed minimum and maximum values. Hence, the attribute **Type** is mandatory, while **Minimum** and **Maximum** are optional. The value of **Type** must be a valid data type that can be serialized during event dissemination by the middleware. The distance between the field values may be defined optionally using the **Step** attribute to restrict the codomain of possible values even further. If a field is of variable length, a maximum **Size** must be stated to enable deduction of the total field size. Value fields define all possible **Values** of a certain **Type** explicitly and do not support additional codomain restrictions or size indications.

If the list of fields is empty or no **Payload** is specified at all, empty messages will be sent.

2) **Classification:** It is required to classify each event type according to the multidimensional classification schema defined in the domain profile. An event type is classified by denominating dimension classes which correspond to the requirements of the event type.

The **Classification** is specified in form of a list of mappings. Each mapping key must reference a dimension instance from the domain profile as defined in **Middleware.Domain.DimensionDefinitions**. Each corresponding value must be an identifier of a class of this dimension defined in the **Classes** attribute of the respective **Dimension instance**.

3) **Explicit Parameter Assignments:** In addition to the configuration parameter values derived from the domain profile, developers can state explicit parameter assignments or preset references to influence the configuration process using the **Parameters** attribute.

Scalar assignment are represented as a list of mappings named **Scalars**. Each mapping key must identify a parameter from the domain profile that has been previously defined in **Middleware.Domain.ParameterDefinitions**. **Scalars**, while each value must be a value of the codomain of the parameter.

![Structure of Event Type descriptions](image)

Fig. 8. Structure of Event Type descriptions
Parameter presets are referenced as a list of identifiers. Each list entry must correspond to the identifier of a parameter preset that is defined in Middleware.Domain.ParameterDefinitions.Presets.

4) QoS Requirements: To automate the decision for a QoS-optimal configuration, application developers must state which limits of QoS-metrics might not be exceeded to ensure sufficient application performance. Tolerable limits are gathered under the QoS attribute.

For each QoSRequirement the Minimum and Maximum values for the corresponding metric are mandatory. Additionally the Correlation between metric and application performance must be defined to provide an optimization target during configuration. Positive correlation means that the performance increases with the values of the metric, while negative correlation means that increasing metric values lead to a decreased performance. No other values are permitted for the Correlation attribute.

Example
We conclude the detailed language specification of MATINEE with an example of an event type description, as shown in Listing 8. The exemplary description models the event type Movement that represents the majority of events in an MMOG. It is used to distribute information about the movement of avatars in the virtual world.

We specified the codomains for the different payload fields. To allow a meaningful interpretation of the events, it is required to disseminate the identifier of the entity to be moved, as well as its updated coordinates in the virtual world.

The multidimensional classification of the event type follows the exemplary schema previously defined in the corresponding domain profile in section IV-C. No explicit parameter assignments or references to presets are required for this event type.

The QoS requirements are defined in order to allow the decision for an optimal configuration. A configuration must guarantee an average latency below 300 milliseconds and an average event loss below 10 percent.

--- !EventType

Metadata:
Identifier: Movement
Description: Player movement in the virtual environment.
Author: Andreas M. Wahl
Date: 08-15-2013
Version: 1.0

Payload:

- EntityID:
  Type: int
- PositionX:
  Type: float
  Minimum: 0.0
  Maximum: 5000.0
  Step: 0.1
- PositionY:
  Type: float
  Minimum: 0.0
  Maximum: 5000.0
  Step: 0.1

Classification:
- Context: Large-scale movement-like
- Validity: Short-lived timer
- Synchronization: None

Parameters:
Scalars: []
Presets: []

QoS:
- AverageLatency:
  Minimum: 0
  Maximum: 100
  Correlation: negative
- AverageLoss:
  Minimum: 0
  Maximum: 0.1
  Correlation: negative

Listing 8. Event Type description for Movement

V. CONCLUSION

The support of application developers is the main purpose of MATINEE. By allowing them to specify the semantics of events in an intuitive way in a domain-specific terminology that does not require deep technical understanding of the middleware and its configuration parameters, MATINEE simplifies the process of QoS-optimal middleware configuration.

In a compact survey of existing languages we have shown that the examined approaches are not suitable for the automated configuration methodology of the M²etis project. We therefore introduced the formal specification of MATINEE, a QoS-aware language for event semantics that is flexible and extensible to be used in arbitrary application domains.

Currently MATINEE is mainly tailored towards the usage as part of the methodology of the M²etis prototype. However, the usage of MATINEE as well as the whole methodology for the configuration of other middleware solutions besides M²etis is certainly feasible and part of our future work. To conclude, the domain-specific expressiveness and extensibility of MATINEE provides an extension to existing QoS modeling languages and enhances the capabilities and scope of QoS modeling towards a more application centric viewpoint.
REFERENCES


