



# **Monitoring Infrastructure (MIS) Software Architecture Document**

**Version 1.1**

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

## Revision History

Date	Version	Description	Author
28-9-2004	1.0	Created	Peter Fennema
8-10-2004	1.1	Processed review comments	Peter Fennema

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

## Table of Contents

1.	Introduction	4
1.1	Scope	4
1.2	Purpose	4
1.3	Overview	4
1.4	Colour convention	4
2.	Identification of stakeholders	4
2.1	Purpose of the system	4
2.2	Stakeholders	4
3.	Monitor example	5
4.	Architectural Goals and Constraints	6
4.1	Goals	6
4.2	Constraints	6
5.	Introduction to views	7
6.	Runtime View	7
6.1	Introduction	7
6.2	Component types and connector types	8
6.3	View description	9
7.	Monitor Design View	10
7.1	Introduction	10
7.2	View description	10
8.	Logical View	10
8.1	Introduction	10
8.2	View description	11
8.3	Package descriptions	11
8.3.1	Events package	11
8.3.2	Monitorclient package	12
9.	Detailed Runtime View	13
9.1	Introduction	13
9.2	View description: Eventrouter package contribution	14
9.3	View description: Events package and monitorclient package contribution	14
10.	References	16

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

## 1. Introduction

### 1.1 Scope

This document applies to the Monitoring Infrastructure (MIS). The MIS is a toolkit for software developers. It provides libraries and tools for local and remote monitoring of events in applications and for graphical representation of those events. It enables visualization of the internal behaviour of technology towards a broad range of audiences.

### 1.2 Purpose

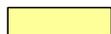
This document provides a comprehensive architectural description of the MIS, using a number of different views to depict different aspects of the system. It intends to capture and convey the significant architectural decisions that have been made on the system.

### 1.3 Overview

Section 2 describes the stakeholders of this architecture. Section 3 gives an example of an application that was built using the MIS. Section 4 describes the architectural goals and constraints. Section 5 gives a short introduction about architectural views. Sections 6 to 9 present the views on the architecture.

### 1.4 Colour convention

To make a clear distinction between the MIS artefacts (libraries and tools) and the context in which those artefacts are used (domain specific applications) the following colour convention is used in all figures of this document:

	MIS artefact
	Domain specific artefact

## 2. Identification of stakeholders

### 2.1 Purpose of the system

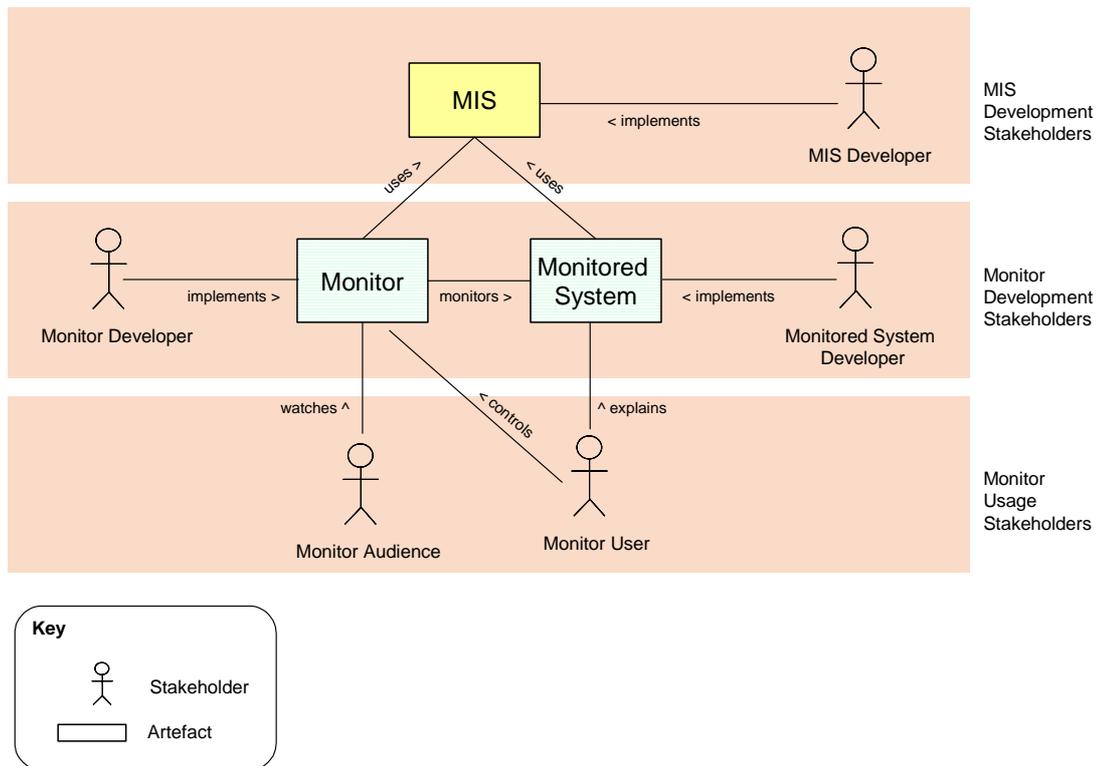
As applications grow more complex in the sense that they involve more distributed components and new technologies or paradigms, it becomes harder to explain the underlying process for demonstration or instruction purposes. A way to gain insight into the behaviour of applications is to visualize the interactions among the different actors or components that are engaged in a process. Seeing the executed process helps to understand how the application works under the hood and what is new or innovative in a particular approach.

The purpose of the MIS is to offer an infrastructure for monitoring events in applications and translation of these events into a graphical representation. The MIS offers libraries and tools that assist developers to implement graphical monitors for their own domain specific applications.

### 2.2 Stakeholders

The stakeholders that have been considered while formulating this architecture are shown in Figure 1. Stakeholders can be categorized based on their activities with respect to the MIS (shown as layers in the figure).

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004



**Figure 1 Stakeholders**

MIS development stakeholders

The *MIS Developer* is responsible for development and maintenance of the infrastructure. He creates a set of tools and libraries that capture the generic behaviour of (graphical) application monitoring.

Monitor Development stakeholders

The Monitor Development stakeholders are the direct users of the MIS as an end product. They are software developers and they use the components and libraries of the MIS to implement monitors for their own domain specific applications. They use the MIS at design time. To enable external monitoring an application has to produce events that reflect its state. The *Monitored System Developer* instruments his application by inserting code that produces events. The *Monitor Developer* develops a graphical representation of the system and implements the logic that receives and processes the events. Monitor Development stakeholders want to focus on their domain specific issues.

Monitor usage stakeholders

The *Monitor User* uses a monitor to explain an application to the *Monitor Audience*. The Monitor User uses the MIS at runtime without being aware of the existence of the MIS at all. The Monitor User is only aware of the Monitor and the Monitored System that were created by the Monitor Development stakeholders.

**3. Monitor example**

Figure 2 shows an example of a Monitor application in a web browser. The Monitor represents a platform (Monitored System) for the distribution of high-quality digital content from and to the home environment. Moving yellow rectangles represent the messages that are flowing around in the platform, illustrating the system dynamics. The Monitor User can explain the platform to the Monitor Audience. He can use the timing controls to tweak the speed of the Monitor or use the step-mode to go through the events step-by-step. Note that this monitor can switch between various views that are targeted towards different kinds of

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

audiences. It also contains a simulator that can play various predefined scenarios.

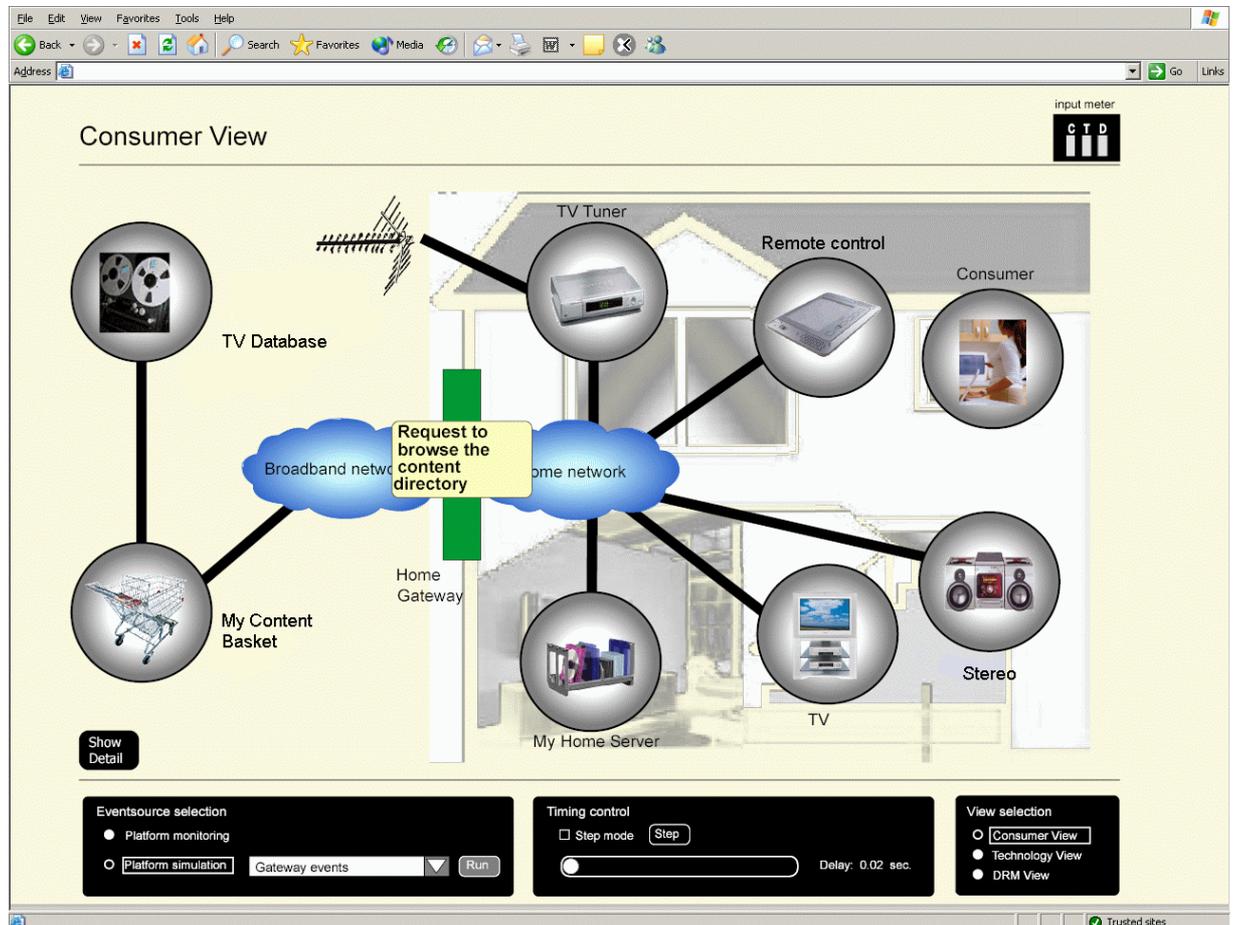


Figure 2 Monitor example

## 4. Architectural Goals and Constraints

### 4.1 Goals

Main goal of this architecture is to serve as a means for education and communication. The MIS is an open source project. The architecture is used to introduce new people to the system. The people may be new members of the development team (MIS development stakeholders) or Monitor Development stakeholders who download the open source distribution. This document will be available on the MIS open source project website [MIS SF].

### 4.2 Constraints

The following key requirements and system constraints have a significant bearing on the architecture of the MIS:

- The MIS is an open source project under the GNU LPGL license [GNU]. This has impact on decisions with respect to inclusion and redistribution of external software libraries or components. All relevant licenses have to be studied carefully.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

- The MIS is intended for instruction purposes only. It will not support high reliability or industry strength monitoring of mission critical processes.
- The MIS must be easy to use with respect to development and deployment of monitors and monitored systems. It must facilitate the needs of Monitor Development stakeholders.

## 5. Introduction to views

A view is a representation of a set of system elements and the relationships associated with them. Different views support different goals and uses. Because the MIS is a toolkit for software developers both the architecture of domain specific applications that use the MIS and the architecture of the MIS itself are relevant. The Runtime View (section 6) describes the architecture of domain specific applications developed by Monitor Development stakeholders. As such it provides information to the MIS Developer about the context in which the MIS artefacts will be used. The Monitor Design View (section 7) describes the architecture of the design tools that are offered to the Monitor Developer. The Logical View (section 8) focuses on the architecture of the MIS itself. It describes the decomposition of the system in packages as well as the behaviour of the system. The Detailed Runtime View (section 9) shows how the packages specified in the Logical View contribute to the architecture of domain specific applications in the Runtime View.

## 6. Runtime View

### 6.1 Introduction

The Runtime View of the architecture describes the runtime components and their interactions. The Runtime View is based on the Component and Connector view type as described in [PCL]. It is similar to the Structural Viewpoint described in [IEEE-1471].

It is important to realise that the delivered artefacts of the Monitoring Infrastructure are software libraries and development tools. The Runtime View in this section does not describe those libraries and tools, but intends to show the context in which they will be used and deployed by Monitor Development stakeholders. The relation between the components shown in the Runtime View and the MIS artefacts is described in section 9.

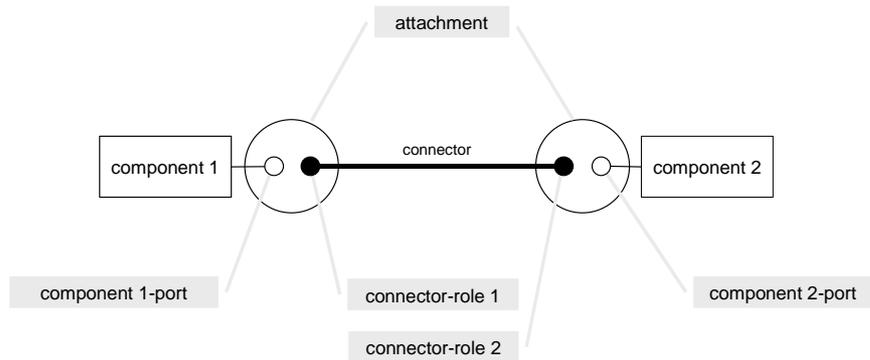
The Runtime View is relevant for:

- The MIS Developer as a guideline for software design and implementation. MIS Developers have to be aware of the application context of the libraries in order to optimise the usability.
- The Monitor Developer and Monitored System Developer, because it shows the various component interconnection possibilities that can be applied in their monitor applications at runtime.

Figure 3 shows the concepts of the Component and Connector View, as they are defined in [PCL].

The connector is an instance of a connector type and each component is an instance of a component type. A connector type represents a form of interaction between components. A component has (one or more) ports (interfaces). Connectors are characterized by their roles. A role can be thought of as the interface of a connector. It defines the way in which components may use the connector to carry out interactions. An attachment attaches the ports of component instances to roles of connector instances.

With respect to this document the abstraction level of component ports connecting to connector roles at attachment points is too detailed. The connector role concept will not be used. In this document component ports will be directly attached to connectors.



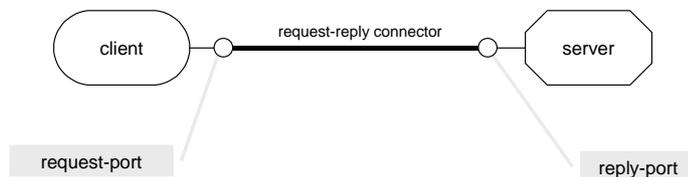
**Figure 3 Component and Connector View concepts**

Section 6.2 describes the component types and connector types that will be used in the presentation of the Runtime View. The Runtime View itself is provided in section 6.3.

## 6.2 Component types and connector types

### Client and server component types with a request-reply connector type

Figure 4 shows how a client and a server component are connected using a request-reply connector. The request-port of the client is attached to the request-reply connector. A server has a reply-port. The reply-port is connected to the other side of the request-reply connector. The client sends events over its request port to the server. The event description is a parameter of the request. The server processes the event and replies with an appropriate message to its reply-port.



**Figure 4 Client server connection over a request-reply connector**

### Publisher and subscriber component types with a publish-subscribe connector type

Figure 5 shows how a publisher and a subscriber component are connected using a publish-subscribe connector.



**Figure 5 Publisher subscriber connection over a publish-subscribe connector**

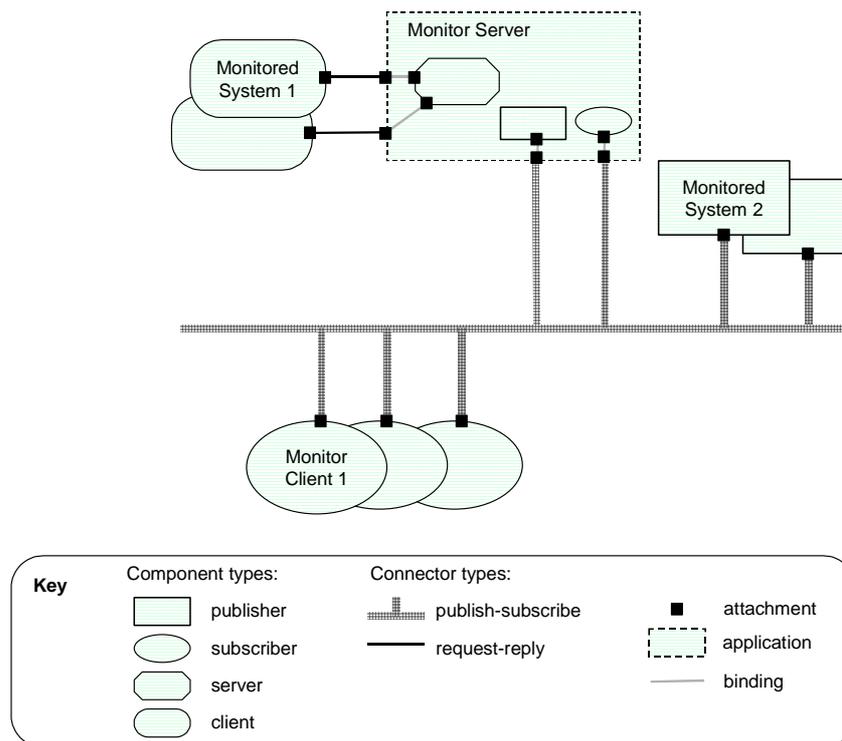
A publisher has a publication-port. The publication-port is attached to publish-subscribe-connector. A

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

subscriber has a subscription-port. The subscription-port is connected to the connector. The publisher sends events to its publication-port and the subscriber receives the event. The publisher is unaware of the subscriber. The publish-subscribe connector decouples subscribers from publishers.

### 6.3 View description

Figure 6 shows the Runtime View. It shows components that pass events, either through a publish-subscribe connector or through a request-reply connector. This figure is one example out of many possible configurations of components and connectors that Monitor Development stakeholders might apply. The example covers most possible usage scenarios.



**Figure 6 Runtime View**

The *Monitor Server* in the figure is a domain specific server application that uses the MIS libraries. It is an example of an application that is developed by a Monitor Developer. It manipulates incoming events (e.g. filtering, queuing) before publishing them to the publish-subscribe connector. The Monitor Server can receive events over a request-reply connector (in this case it acts as a server component) or over a publish-subscribe connector (in this case it acts as a subscriber component). The Monitor Server can also publish events over a publish-subscribe connector thereby acting as a publisher component.

*Monitored System 1* is a domain specific application that uses the MIS libraries to send events to a Monitor Server over a request-reply connector (acting as a client component).

*Monitored System 2* is a domain specific application that publishes events directly to the publish-subscribe connector without any intermediate manipulation (acting as a publisher component).

The *Monitor Client* is connected to the publish-subscribe connector. It is acting as a subscriber component. It receives the events published by the other components and typically will show graphical updates in a web browser.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

## 7. Monitor Design View

### 7.1 Introduction

The Monitor Design View describes the design tool (Monitor Client Design Tool) that the MIS offers to the Monitor Developer. The view is based on the Pipe-and-Filter Style as described in [PCL].

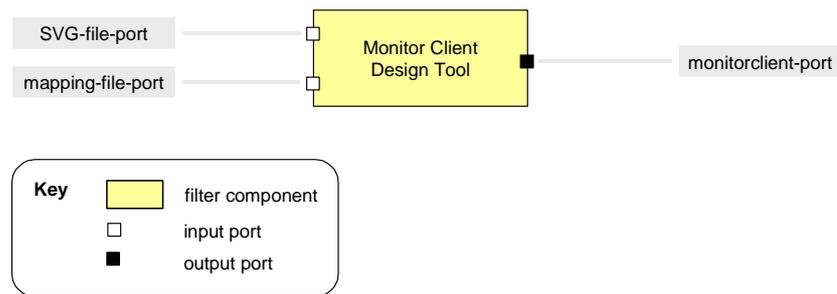
The Monitor Design View is relevant for:

- The MIS Developer as a guideline for software design and implementation of the Monitor Client Design Tool.
- The Monitor Developer as a functional description and a usage guide of the Monitor Client Design Tool.

### 7.2 View description

The Monitor Client Design Tool addresses the problem of translation of incoming events into graphical updates. Complex applications will often require complex visualizations (with a large number of graphical objects) and generate many types of events. Programming this many to many mapping can be a tedious job. The Monitor Client Design Tool supports the Monitor Developer at design time by generating all mapping logic based on a mapping that is defined in a XML file.

The Monitor Client Design Tool is best represented as a filter that transforms input files into a dedicated package (monitorclient package). This package can subsequently be deployed in a web browser environment. This is shown in Figure 7.



**Figure 7 Monitor Design View**

The Monitor Client Design Tool is executed from the command line. On its SVG-file input port it expects an SVG file that contains the graphical representation of the Monitored System. SVG is a XML based W3C standard that is programmable through java or JavaScript and is very extensive in terms of describing graphical concepts [SVG]. On the mapping-file input port the Monitor Client Design Tool expects an XML file that represents the many-to-many mapping of events to JavaScript function calls. These function calls will trigger graphical updates of the SVG document at runtime. On the monitorclient output port the Monitor Client Design Tool will create a monitorclient package that contains all mapping logic and graphics in an easy to deploy format. The structure of this generated monitorclient package is explained in section 8.3.2.

## 8. Logical View

### 8.1 Introduction

The logical view of the architecture describes the decomposition of the system into packages. For each relevant package it introduces the architecturally significant classes and describes their responsibilities, as

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

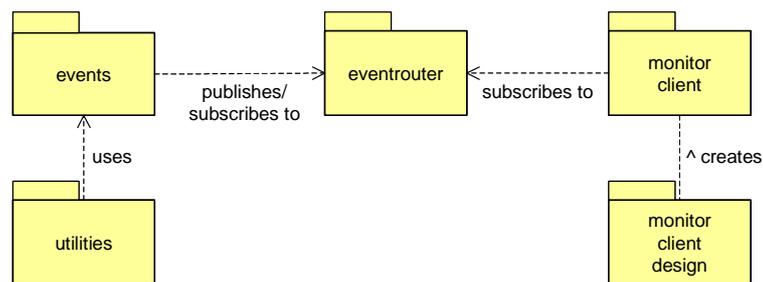
well as a few very important relationships, operations, and attributes. The logical view is based on the logical viewpoint as described by the Rational Unified Process [RUP].

The logical view is especially relevant for the MIS Developer as a guideline for software design and implementation.

Section 8.2 describes the Logical View and section 8.3 describes some of the packages in more detail.

## 8.2 View description

The logical view of the MIS is shown in Figure 8.



**Figure 8 Logical View**

The *events* package is a software library that will be used by Monitor Developers. They will integrate this library into their applications, both for Monitors and Monitored Systems. It has classes for event publishing, event delaying, queuing and event filtering. The events package is explored in more detail in section 8.3.1.

The *utilities* package is a software library that offers a simulator that can publish events that are specified in a play list. The Monitor Developer can use this package at design time for testing purposes. The Monitor Developer can also include a simulator in the final Monitor, to enable the Monitor User to show predefined event sequences to the Monitor Audience. The play list is defined in a dedicated XML file format.

The *eventrouter* package is a middleware package for event routing. The eventrouter decouples event publishers from event subscribers. The eventrouter uses the concept of topics. Applications can publish events to topics on the eventrouter. Clients can subscribe to topics, and receive the events that are published towards those topics. The eventrouter package is a ready-to-deploy web application package that is used by Monitor Developers. An off-the-shelf component is used for the eventrouter: the pushlet framework [PUSHLET]. The internals of the eventrouter are not architecturally significant. The runtime interface of the eventrouter is described in section 9.2.

The *monitorclientdesign* package represents the Monitor Client Design Tool as described in section 7. The Monitor Developer uses this tool at design time. It is a command line tool for code generation that is not used as part of a Monitor at runtime.

The *monitorclient* package is the bundled result of code generation with the Monitor Client Design Tool. The Monitor Developer will in most cases integrate this logic in a Monitor Client. The monitorclient package is explored in more detail in section 8.3.2.

## 8.3 Package descriptions

### 8.3.1 Events package

Figure 9 shows the most important classes of this package.

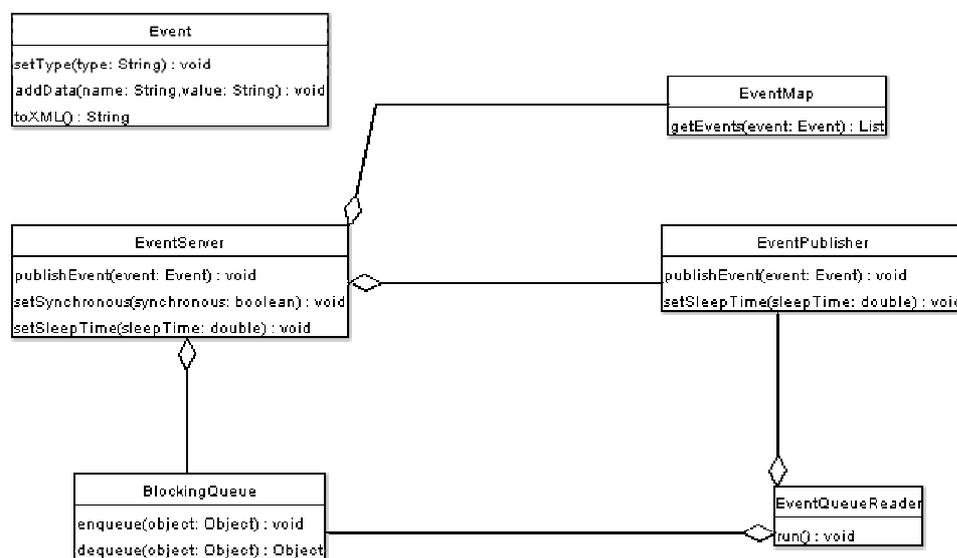
The *Event* class models an event. An event has a type, and it is possible to attach data to an event. Event data are passed as name-value pair strings. An event can be serialised to XML.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

An *EventServer* is the concept that clients use to publish events. The actual event publication is either directly delegated to an *EventPublisher* or stored in a *BlockingQueue* for intermediate event storage. This depends on the settings of the “synchronous” parameter. If synchronous event publication is desired the thread that calls the *EventServer* for publishing events is blocked until the event is really published. In case of asynchronous event publication the thread returns immediately. The events are stored in the *BlockingQueue*. Another behavioural parameter is the “sleeptime”. Specifying a sleeptime will hold publication of the event. The event is only published after the sleeptime has elapsed. This can be used to delay the event flow to a *Monitor* in the case that the speed of subsequent events is so high that the resulting graphical updates in the *Monitor* are too fast for perception by human beings. The *EventServer* can be configured with a URL and a topic, representing the destination of the event.

The *EventPublisher* does the actual publication of the events. It can handle various types of interfaces. The description of the runtime interface and connection protocols is given in section 9.3.

The *EventQueueReader* retrieves the events from the *BlockingQueue* and calls the *EventPublisher* to publish the event. The speed of emptying the queue is determined by the sleep time.



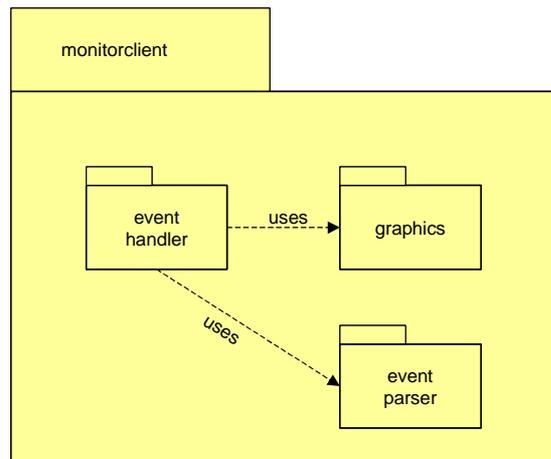
**Figure 9 Architecturally relevant classes for event package**

The *EventMap* specifies how to handle incoming events. An *EventMap* can be registered with an *EventServer*. It has an entry that describes criteria for incoming events and for each entry it has a specification of zero or more outgoing events. Before publishing an event the *EventServer* compares this event with the criteria for incoming events in the *EventMap*. If a match is found the outgoing events as described in the *EventMap* will be published instead of the original event. This enables modification of parameters of incoming events before publishing them, or inserting additional events, or filtering out events. The specification of event mappings is read from a XML based file format.

### 8.3.2 Monitorclient package

Figure 10 shows the Logical View of the monitorclient package.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004



**Figure 10 Logical View, monitorclient package**

The monitorclient package can be decomposed into 3 sub packages:

The *graphics* package contains the graphical representation of the Monitored System in SVG format [SVG]. It also has an API to manipulate those graphics (e.g. colour and shape updates, starting and stopping of animations). The structure of the graphics package is generated at design time by the Monitor Client Design Tool and is therefore domain independent. The SVG graphics are domain specific. The API is for a large part determined by the SVG standard, but it can be extended with user-defined functions.

The *eventparser* package parses the incoming events. An off-the-shelf component is used for the eventparser: XML for Script [XFS].

The *eventhandler* package contains the logic that maps incoming events to function calls of the API of the graphics package. The eventhandler logic was generated at design time with the Monitor Client Design Tool. The structure and the interface of the eventhandler package are domain independent. The implementation is domain specific.

The monitorclient package is intended to run in a web browser environment using JavaScript as scripting language.

## 9. Detailed Runtime View

### 9.1 Introduction

The Detailed Runtime View zooms in on the Runtime View of section 6. It shows how the packages specified in the Logical View contribute to the architecture of domain specific applications. It consists of 2 separate sections: section 9.2 describes how the Event Router package of the Logical View contributes to the publish-subscribe connector and section 9.3 will show how the events package and the monitorclient package of the Logical View contribute to the publisher, subscriber, client and server components of the Runtime View.

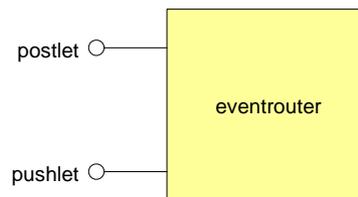
The Runtime View is relevant for:

- The MIS Developer as a guideline for software design and implementation. MIS Developers have to be aware of the application context of the libraries in order to optimise the usability.
- The Monitor Developer and Monitored System Developer, because it shows the various component interconnection possibilities that can be applied in their monitor applications at runtime.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

## 9.2 View description: Eventrouter package contribution

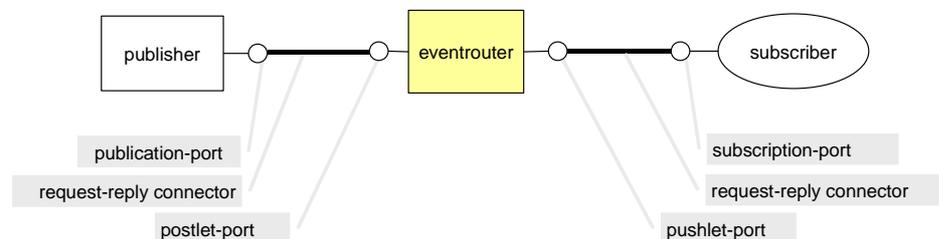
Figure 5 and Figure 6 use a publish-subscribe connector as an abstraction. The Logical View mentions a concrete eventrouter package that realises this abstract connector. The Detailed Runtime View replaces the representation of the publish-subscribe connector in Figure 5 by the runtime representation of the eventrouter package and the required connectors. Therefore it is necessary to understand the runtime interface of the eventrouter. The runtime representation of the eventrouter is shown in Figure 11.



**Figure 11 Runtime representation of the pushlet event router**

The eventrouter has 2 runtime interface ports. The *postlet* port is used for event publication. The *pushlet* port is used for subscription to events. Clients communicate with the eventrouter by means of HTTP request with a specific parameter format. Clients that want to *publish* events can send a HTTP request to the *postlet* port. Clients can *subscribe* to events by sending a request to the *pushlet* port. After receiving the request for subscription on the pushlet port the eventrouter does not close the connection but keeps it open and pushes fresh events to the client. This is a technique similar to HTTP streaming as for example is used in multimedia viewing applications. For a more detailed explanation and interface specification see [PUSHLET].

The eventrouter contribution to the Detailed Runtime View is shown in Figure 12 (compare this figure with Figure 5).



**Figure 12: Detailed Runtime View, eventrouter package contribution**

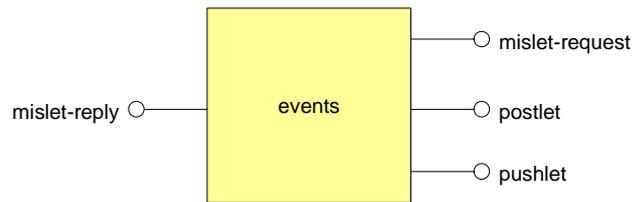
The publication-port of the publisher is connected to the postlet-port of the eventrouter by a request-reply connector. The subscription port of the subscriber is connected to the pushlet-port of the eventrouter by a request-reply connector.

## 9.3 View description: Events package and monitorclient package contribution

To express the contribution of the events package and the monitorclient package to the Detailed Runtime View it is necessary to have the runtime representation of those components.

The events package as described in the logical view directly maps to a runtime component. The runtime representation of the events package is shown in Figure 13.

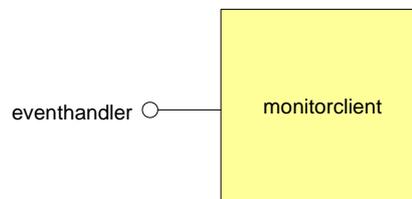
Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004



**Figure 13 Runtime representation of the events package**

The events package has 4 ports. The pushlet port can be used to subscribe to an event router. The postlet port can be used to send events to an event router. The mislet-request port can be used to call a server over a request-reply (in this case HTTP) connector where the event is passed as a collection of parameters in the so-called *misset-format*. The mislet-reply port can receive events in the mislet format.

The monitorclient package as described in the logical view also maps to a runtime component. The runtime representation of the monitorclient package is shown in Figure 14.



**Figure 14 Runtime representation of the monitorclient package**

The only port of the monitorclient package is the eventhandler port. This port can receive incoming events.

The Detailed Runtime View is best described by adding the runtime representation of the events package and the monitorclient package to the Runtime View of Figure 6. The result is shown in Figure 15.

The *Monitor Server* uses the events package for multiple purposes. It can receive events over a request-reply connector at the mislet-reply port. It can subscribe to events using the pushlet port. It can publish events to the publish-subscribe connector using the postlet port.

*Monitored System 1* uses the events package to send events to the mislet-request port. This port is connected to the mislet-reply port of the Monitor Server.

*Monitored System 2* uses the events package to publish events to the publish-subscribe connector using the postlet port.

The *Monitor Client* is connected to the publish-subscribe connector. It is acting as a subscriber component. It receives the events published by the other components. It calls the eventhandler port of the monitorclient package to parse the incoming event and to show graphical updates in the web browser.

The example in Figure 15 illustrates how Monitor Development stakeholders can use the events package and the monitor package in their applications. The packages handle all distributed communication and timing issues, so the developers only have to concentrate on the domain specific aspects of their monitoring applications.

Monitoring Infrastructure (MIS)	Version: 1.1
Software Architecture Document	Date: 8-10-2004

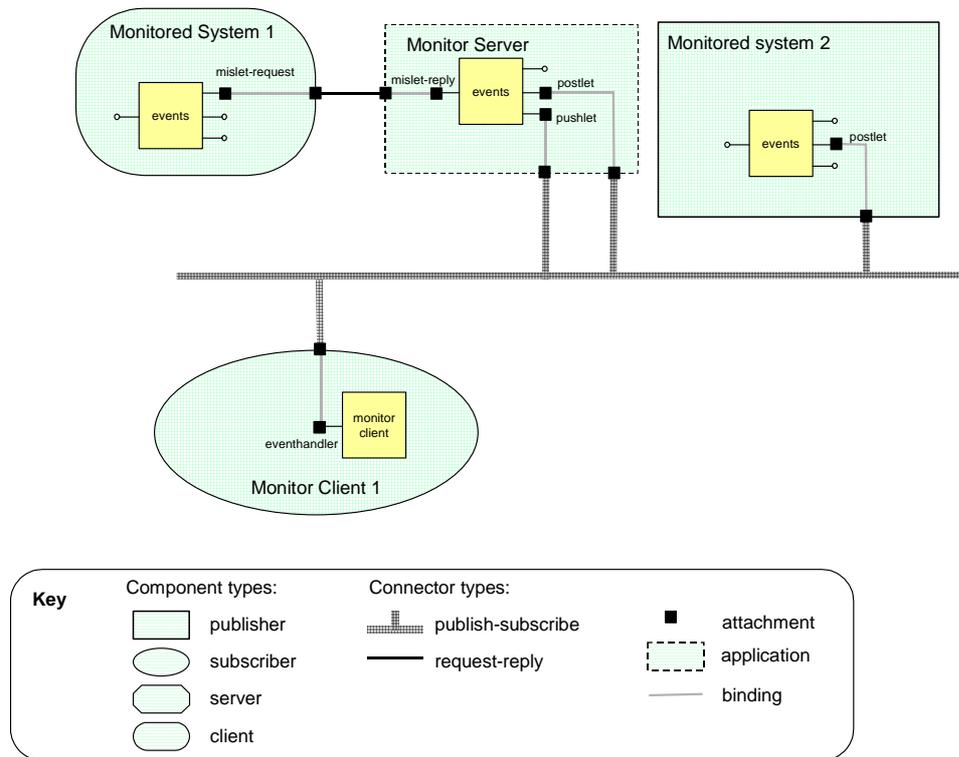


Figure 15: Detailed Runtime View, events and monitorclient packages contribution

## 10. References

- [MIS SF] MIS project website at SourceForge, <http://mis.sf.net>.
- [GNU] GNU LGPL license, <http://www.gnu.org/copyleft/lesser.html>.
- [PCL] P. Clements, F. Bachman et al., *Documenting Software Architectures*, Addison-Wesley 2003.
- [IEEE-1471] B. Sherlund et al., *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 2000.
- [RUP] Rational Unified Process, <http://www-306.ibm.com/software/awdtools/rup/>
- [PUSHLET] The pushlet framework, <http://www.pushlets.com/>
- [SVG] Scalable Vector Graphics, <http://www.w3.org/Graphics/SVG/>
- [XFS] XML for Script, <http://xmljs.sourceforge.net/index.html>