

An UML Profile for Tivoli Event Management Modeling

Draft Study

Author: Massimo Coletti

Director Information Technology Area

Version 1.1 released on Aug. 21, 2003



BANCA FINNAT EURAMERICA

1. Abstract

Management of Enterprise Resources using a sophisticated tools suite like Tivoli is a complex activity, and requires a careful analysis.

Our Company is using Tivoli as a tool for comprehensive management of Business Applications, providing our service personnel with selected alerts and automated support when an application is not working properly.

This paper describes an UML profile designed to support our Analysts in the Analysis and Modeling activities required prior the implementation of a monitoring and event management solution.

This paper applies to Tivoli Enterprise console 3.8 and UML 1.5.

Copyright Notice

Copyright © 2003 – Banca finnat Euramerica S.p.A.

The material included in this paper is property of Banca finnat Euramerica S.p.A. - Rome Italy.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	OS/390	Tivoli Enterprise Console
IBM	Tivoli	TME
Tivoli Logo	IBM Logo	
OpenEdition	Tivoli Enterprise	

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.



Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

2. Definitions

To define a standard modeling structure, we need to identify a finite number of elements to be represented in our UML profile.

Tivoli Elements

The elements specific of the Tivoli solution are the following:

- Tivoli nodes (Endpoints, Gateways, Managed Nodes, Enterprise Consoles, etc.);
- Tivoli Events sources;
- Tivoli Events;
- Tivoli Events definitions (BAROC files);
- Tivoli Monitors, sources of Tivoli Events (elements of Tivoli Monitoring or TEC Adapters);
- Tivoli actions (events processing or task execution).

Business Application Elements

The elements representing a Business Application are:

- hosts (or physical nodes);
- application components, usually defined as “servers”, offers ports to other application components.

3. Profile Stereotypes

The following table identifies the stereotypes used to model the different elements which constitutes the profile, and the base class to which the stereotype may be applied.

Stereotype	Base Class
TME Endpoint	Node
TEC Server	Component
TEC Gateway	Component
TME Managed Node	Node
Event Source	Package
Event	Class
BAROC (Event definitions)	Package
ITM Monitor	Class
TEC Adapter	Class
NW Polling	Class
Rule	Class
Task	Class
Host	Node
Network interface	Interface
Server	Component
port	Association
raise	Association
monitors	Association
clears	Association (trace)
acknowledge	Association (trace)

Tagged values

The following tagged values are defined in this profile:

Tagged Value	Applies to	Allowed values
IsClearingEvent	Event	true

4. Modeling methodology

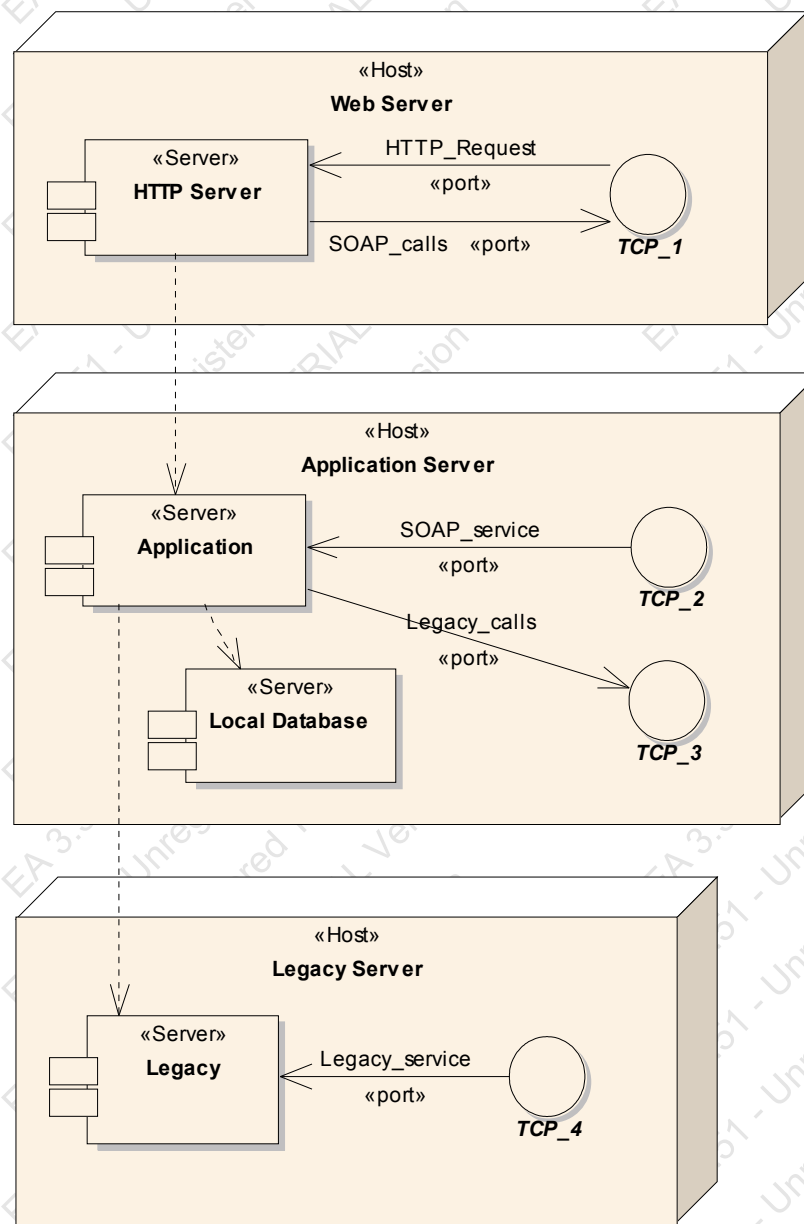
Model the application

Modeling an application require to identify the application components, the protocols used for inter-component communication, and the physical nodes and resources used for deployment.

A second step in application modeling is discovering the functional dependencies among components, that means finding, for each component, the other components that should work in order to ensure the proper component behaviour.

In term of UML syntax, this implies the design of a Deployment diagram.

Application Model Example



The above diagram shows an application distributed over three hosts.

For each host, are shown the application components deployed on it.

Constraint: an application component may be deployed only on one host.

The diagram shows how each component rely on the other for its proper operation through a *dependency* association.

In the example, we see that the **Application** could work only if **Local database** and **Legacy** are working.

If the **HTTP Server** is down, the rest of the application continues to work (nevertheless the users won't be able to access the website).

The diagram shows also the network interfaces of each host, and the services provided or requested through them. This element is important if the Tivoli environments includes NetView or other means to monitor the network resources at TCP/IP level.

The dependency of the application components to the network interfaces are not explicit, but should be taken into account.

Identify monitoring points

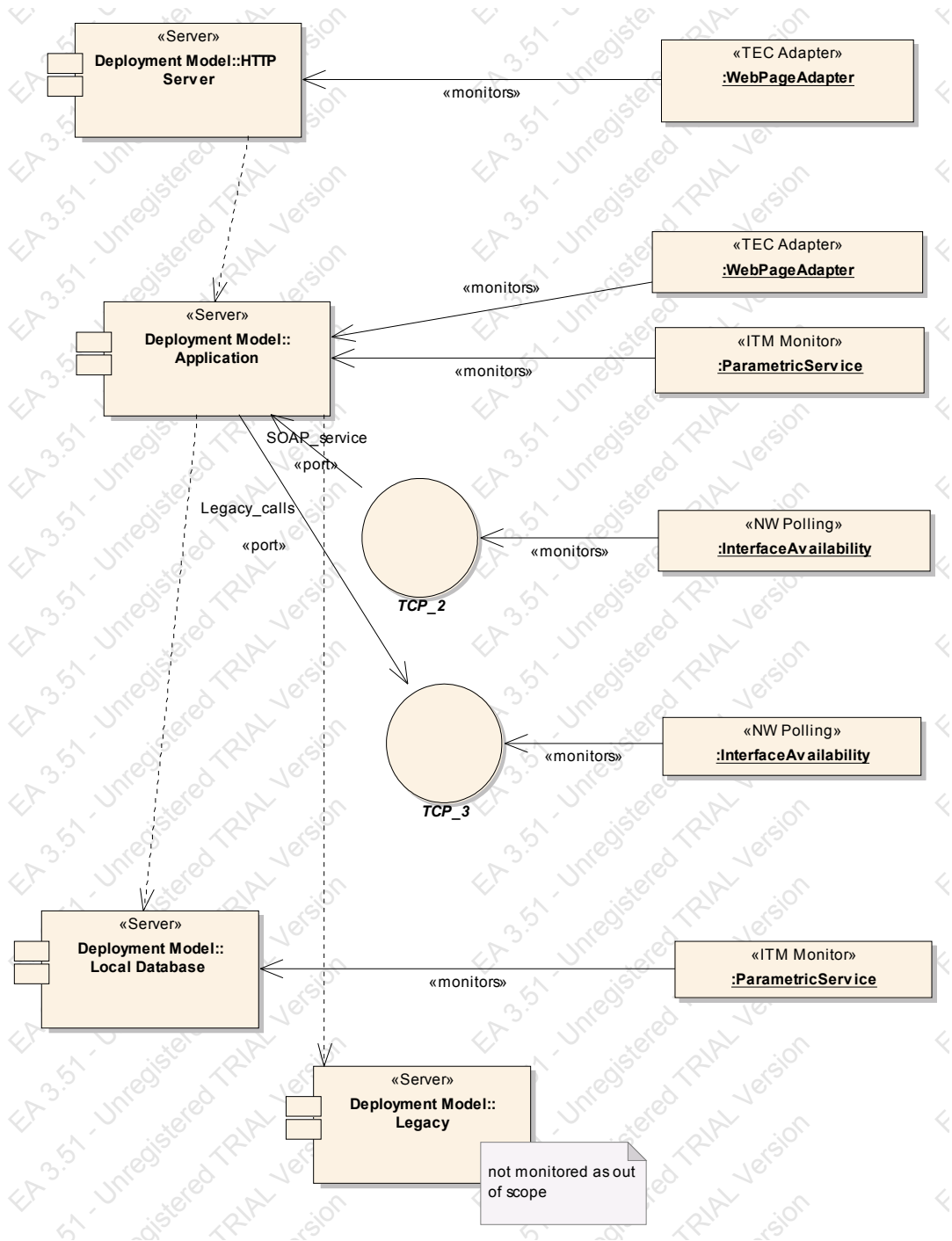
This phase requires to identify all the significant monitors available for each application component, including the interfaces.

All the relevant monitors should belong to an Event Source, modeled with a package including both the monitors, the events, and the event definitions. The Event source is defined at the Enterprise Console level.

Each monitor should be associated with the monitored component.

On the following page, you can see an example.

Example of monitoring points model



In the example all of the application components are shown.

An **instance** of each monitor is shown, with its relationship with the monitored components.

Identify raised events

Each monitor class should be analysed in order to enumerate all the possibly raised events.

This relationship can be represented by an association between the monitor and the event class.

This step, together with the previous one, allow us to build a matrix Application Component/Event.

For each event, all the relevant attributes should be identified, and shown as public attributes in the class definition. This analysis should provide all the relevant input for the .baroc file designer, as well as for the monitor implementor.

Clearing events may be identified with a tag *isClearingEvent*.

The diagram should map the relationship among clearing and cause events with a trace association with stereotype «clears» from the clearing event to the cleared one. The analyst may use source and destination roles to document the tuples of attributes used to link the event together. Different tuple in source and destination identify cases when the slot name is different in the two events.

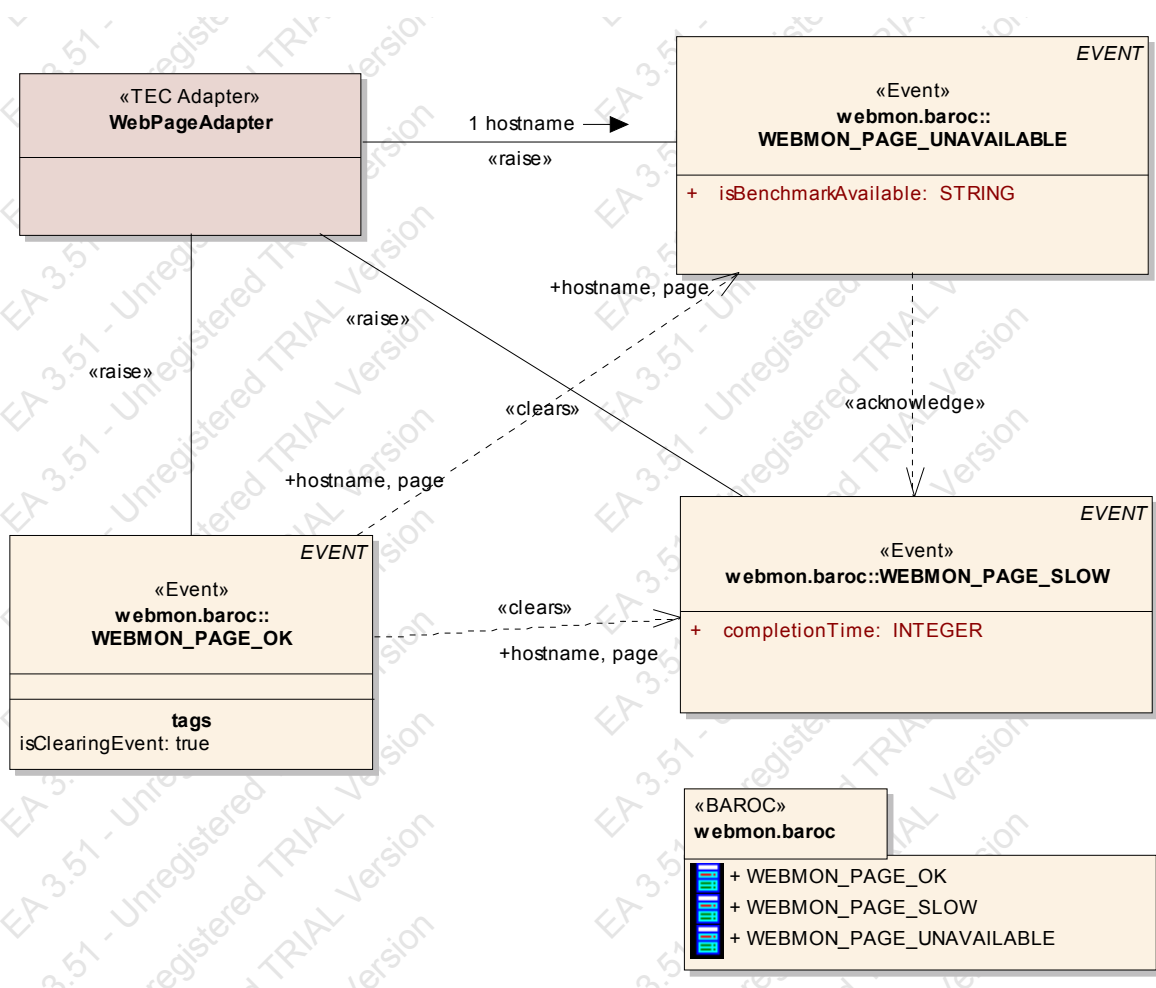
Severity relationships are mapped with trace association with the stereotype «acknowledge» from the highest severity/importance event toward the related ones.

Further documentation should include an analysis of the component state change triggering the event generation. This analysis helps identifying the proper severity level associated to each event.

Example of Event source package details

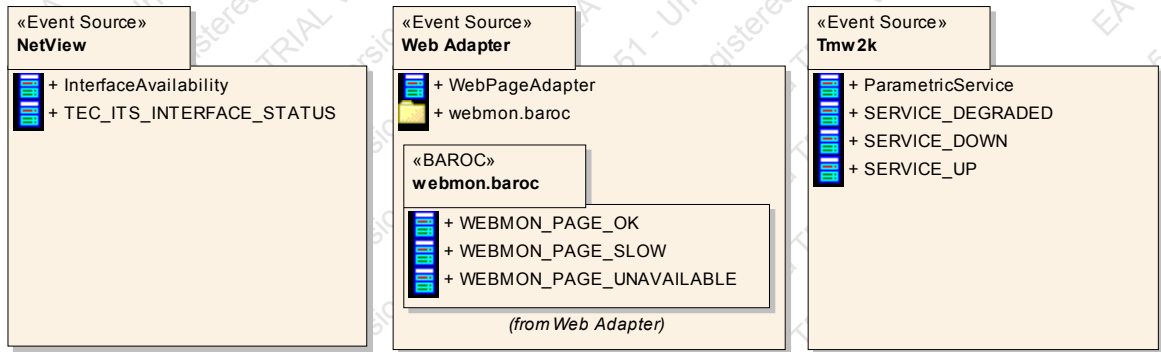
The following diagram shows the relationships among monitor and events for a selected Event Source.

The diagram also shows the package associated with the .baroc file/s which contain the event definitions.



In the example is also shown, as a suggestion, that the attribute *hostname* (which is part of the EVENT superclass), is supplied by the monitoring element. This may have the meaning that the attribute content is depending on the particular instance of the monitor, and doesn't vary with different instances of the event.

The next diagram provides with an overview of all the data sources, monitors, and events, involved in this application monitoring analysis:



5. Designing Event Chains

At this point, we are able to represent the chain of events (Tivoli events) originated by a single event affecting the application.

As a starting point, we should try to enumerate the *real world* events that may occur, and then develop all our model's events raised from the real event.

According to the components relationships outlined before, we therefore can identify the dependency relationships among Tivoli Events.

From this diagram we can identify the “cause” Event as should be acknowledged by the event correlation engine.

Another important point is the time sequence of the raised events. A flow association should link the “real” event with the Tivoli Events raised directly from his happening. If the analyst is sure that some events will always arrive to the TEC before others, those event should be linked with a flow association.

The resulting diagram shows, in this way:

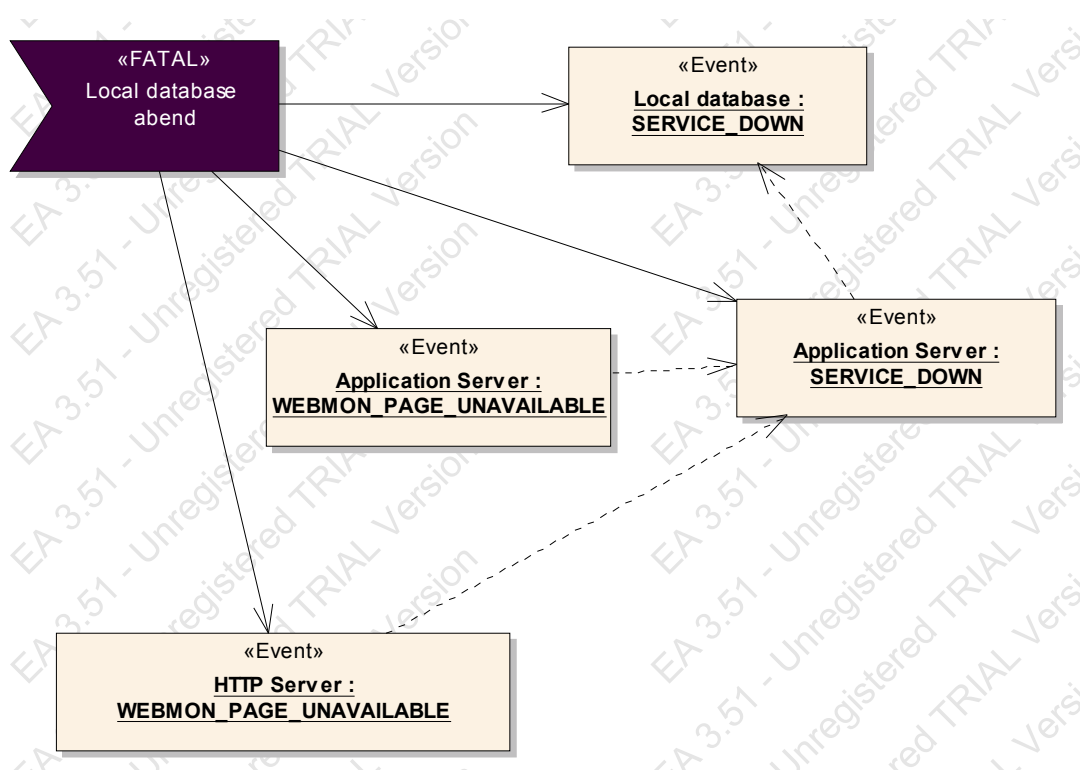
- logical sequences of events (cause/effect);
- time sequences of events.

This information is the base to design `create_event_sequence` rules.

Example of event relationships

The following example illustrate the case of an application abend in a software component.

The abend event causes all the depending components to fail, so we obtain a couple of events:



The originating “physical” event is shown in the upper left corner. The stereotype «FATAL» represents the severity of the event in term of business.

All the raised events are shown as instances of the corresponding classes. The object name is the component name that originates the event. So is possible to distinguish among different events, of the

same class, originated by different components.

The dependency relationships among the Event instances are the same represented in the application model. This diagram doesn't show any defined temporal sequence among events: this means that the events may reach the TEC in any order.

The "cause" effect is SERVICE_DOWN instance from Local Database.

6. Next Steps

A further refinement of the profile is required to support the rules analysis.

Designing an UML profile for Enterprise Architect.

Check overall consistency and traceability.

7. Acknowledgments

This paper was developed as an effort to develop an internal methodology for Event Management design.

The example diagrams were realised with a trial version of Sparx System's Enterprise Architect, a powerful and low-cost modeling tools.

This tool allow the user to develop his own UML profiles, and this is the next development of this effort.

The tool also has a built-in code-generation utility. Implementing this utility with the .barox syntax, could result in an interesting CASE tool for Tivoli Engineers.