

Cockpit based Management Architectures

Robert Neumann¹, Fritz Zbrog¹, Reiner Dumke¹

¹ Otto-von-Guericke University, Department of Distributed Systems, Universitätsplatz 2,
39106 Magdeburg, Germany
{Robert.Neumann, Fritz.Zbrog, Reiner.Dumke}@ovgu.de

Abstract. The following article covers the different types of software cockpit applications for use in various areas, such as software development, software maintenance as well as quality management. Starting with an explanation of the general basics and intentions behind software cockpits, we introduce a dynamic infrastructure that implements dynamic information flows. Furthermore, we will outline which mechanisms this infrastructure provides for the visualization and presentation of those information streams and restrict our elaborations to three independent screens. We will conclude with a discussion of a cockpit prototype that was implemented at our institute.

Keywords: Cockpits, Project Management, Management Architectures.

1 Introduction

Information Technology (IT) is embedded in a complex and dynamic environment. Analyses have shown that IT will develop in line with the increasing globalization of enterprises. Trends indicate that enterprises will have to be reactive and responsive to growing challenges and demands. One of those challenges is the coordination of globally acting teams that are spread across various countries and time zones [1]. One of the results of IT globalization is that enterprises face an increasing cost pressure as well as a high demand for quality and flexibility. In order to fulfill this demand, every IT enterprise requires a well functioning management. Prerequisites for the success of every software project are good project planning, project monitoring and project steering. The management of software projects connects different business areas with each other. Product and process related observations should therefore be adaptable to the various perspectives that are prevalent within organizations. So called “software project control centers” can serve as compact sources of information and help the management to gain a better overview of the current project status.

The current literature uses the terms “control center” and “cockpit” as synonyms. Those provide managers with the possibility to adjust reports about project controlling to questions and interests of individual observers in order to visualize only relevant, controlling related information. The cockpit idea originated from aerospace and aeronautics. A flight, for example, could be considered as a process that is comparable to a development process in IT. Coworkers of different areas participate

in the same process (pilots, air safety, passengers, etc.). Every participant requires different information about the flight (process) or the plane (product) respectively. The passenger on the one hand would like to see information about the flight distance and the time to destination on his private screen. The pilot on the other hand needs to understand the technology, operational reliability and other characteristics of the plane. Lastly, the flight safety is provided with information about the airspace and the planes that are passing through it, in order to be able to coordinate safe routes (Figure 1).



Fig. 1. Cockpit as used in air safety.

In software development, cockpits for the first time allowed for the efficient monitoring of a variety of project related status information as well as the generation of trend analyses. Project information can be displayed in a standardized and transparent manner; the evaluation and assessment of the project is eased significantly resulting in a clear and simple reporting. Thereby, it is clearly not desirable to create new information domains. Instead, existing systems, such as customers, software development and project management are integrated with each other allowing for an enhanced communication between each other. Hence, cockpits can help to establish a more realistic project planning. The complexity and the cost of software development can be reduced to a minimum as process and product related risks can be faster identified.



Fig. 2. Cockpit vision.

Several different approaches support the software development process. Tools and tool infrastructures on the one hand allow for a process planning and controlling that is based on project related measurement and data management (e.g. knowledge plan, Microsoft Project, Function Point Workbench, COCOMO II Tool, Rational and Metrics One, SLIM Tool etc.). The collected data is usually measurement data that reflects the product quality (often based on so called code metrics) as well as project related data extracted from effort estimation and process planning activities (resources, timeline, preconditions).

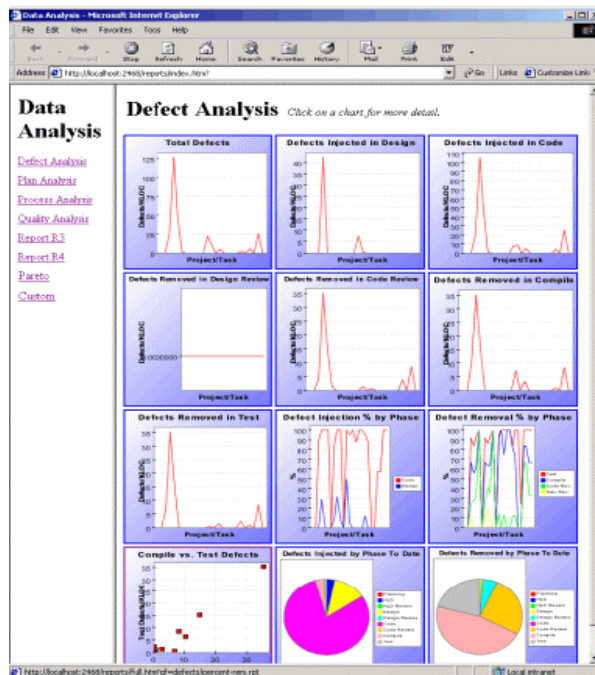


Fig. 3. Example of the adaptation of a cockpit by an enterprise.

Project data for the reporting to the management on the other hand is aggregated and very often processed manually. Figure 3 demonstrates an example featuring an Alcatel product division that introduced a management cockpit in order to support the management activities [3]. In addition to those enterprise specific solutions do certain tendencies aim at supporting decision making through the implementation of more general experiences. The general procedure for the aggregation of project management data is described in the following layered diagram [3].

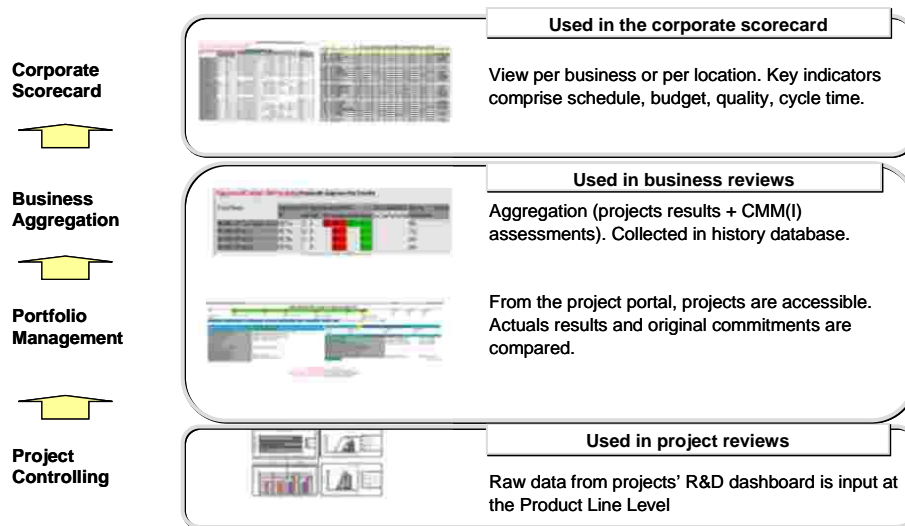


Fig. 4. Cockpit based types of project data aggregation.

2 Basic Cockpit Solution

The analysis of possible representations for process and product data is the starting point for the development of a cockpit solution. By using the cockpit, the management is provided with all the information that is required for an optimal controlling and steering of a software project. In case of a project control center, it is desirable to visualize the individual project drivers on separate screens and in different visualization modes. The measurement, interpretation and representation of the project metrics are realized through tools on the one hand and internet based web services on the other hand. The individual cockpits provide up-to-date and consistent data and hence allow for a well structured access to all relevant information. Furthermore, the current progress of the project is illustrated in a clear and well-arranged manner which supports the easier identification of calls for action.

Based on the measurement and analysis activities such a cockpit incorporates, different methods for the visualization of integrated measurement data exist. Figure 5 demonstrates a visualization mode that combines descriptions of structured and timed activities.

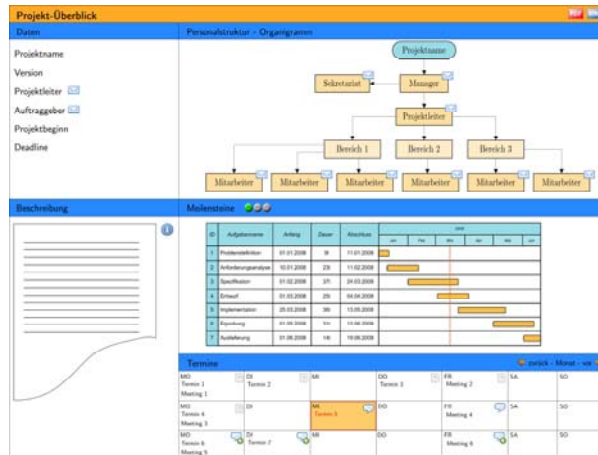


Fig. 5. Cockpit representation for the project management.

The above cockpit allows for the aggregation of project progress data from the individual project metrics in order to establish a novel “project experience database”. One of the outcomes of such a database could be an augmentation of existing measurement possibilities as well as an enhanced precision of the actual results of the analysis (see Figure 6).

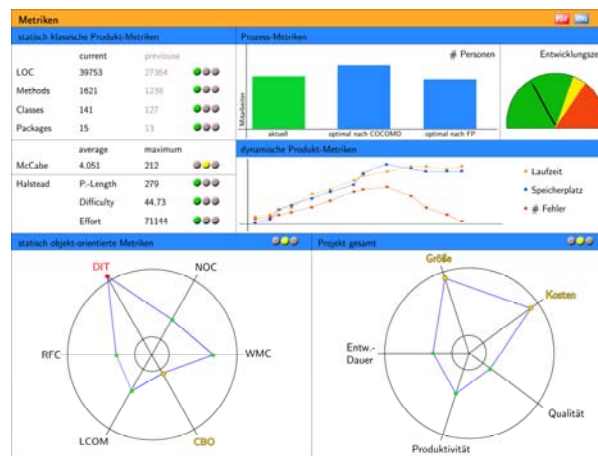


Fig. 6. A detailed representation of product metrics.

In order to evaluate the results that were gained for the control or steering process, complex aggregation and examination operations are applied. Figure 7 depicts a composition that is feasible for the application in project management areas. It is shown how such a composition can be arranged on one screen, whereby only a subset of a much more complex cockpit architecture might be involved.

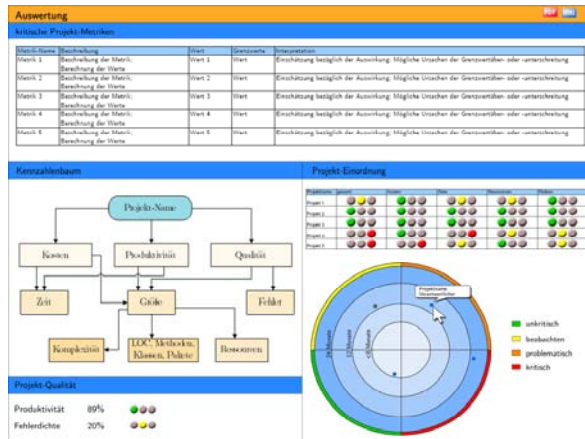


Fig. 7. Analysis of individual project management areas.

The above introduced principles were already applied to an initial prototype of a cockpit solution that is intended to support basic project management activities [4]. Figure 8 demonstrates the Magdeburg cockpit prototype monitoring project management activities.



Fig. 8. Example of a cockpit solution for the monitoring of project management activities.

Our prototype implemented the following two types of infrastructure:

- Two computers support the integration of a screen on the one hand and a multi screen on the other hand
- The applications are separate desktop processes of a single computer

3 Magdeburg's Cockpit Approach

3.1 Architectural Foundations

Regarding our cockpit framework, we formulated the following requirements [2]:

1. Elaboration of construction and development related basics of cockpit engineering for various IT areas
2. Development of an extendable and component-oriented cockpit architecture
3. Evaluation of different technological alternatives for realization
4. Focusing upon a process form that supports capturing of measurement values, processing and visualization of those
5. Development of other experimental prototypes

Regarding the fourth goal, we created the following scenario:

Agent or sensor software acquires the information that is to be represented directly from the source. In the following, a transportation system organizes the consolidation of the information (e.g. data for traffic loads, resource allocation and consumption) in a temporary buffer. After the information went through certain processing and filtering operations, it is going to be visualized. Semi-automatic interaction allows for the embedding of sub systems by the user, whereby the system captures process log data as well as transaction data. Subsequently, the steering and control information that was entered by the user is redistributed via the transportation system back to the agents where the original information came from – the information cycle closes.

Our proposal for a general cockpit architecture looks like the following:

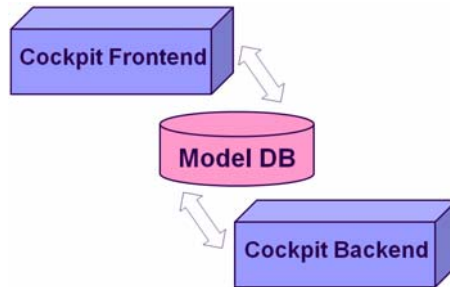


Fig. 9. General cockpit architecture.

The single frontend components of the architecture are represented in the Figure 10 [5].

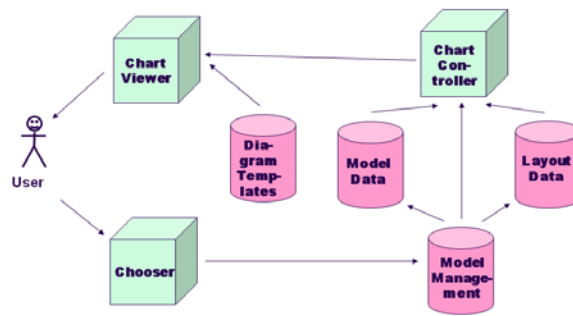


Fig. 10. Frontend components of a cockpit.

The components of the cockpit backend are shown in Figure 11 [5].

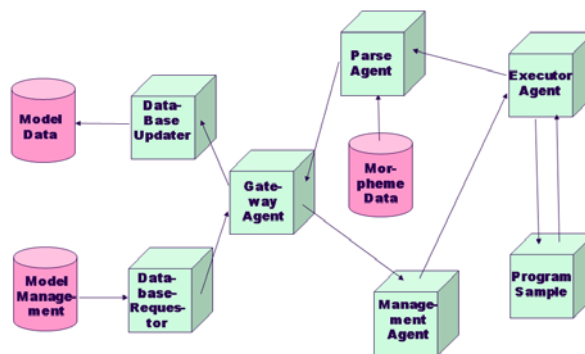


Fig. 11. Backend components of a cockpit.

3.2 Realization Approaches for the Cockpit Layer Architecture

From the earlier mentioned information cycle we extracted the following general cockpit layer architecture.

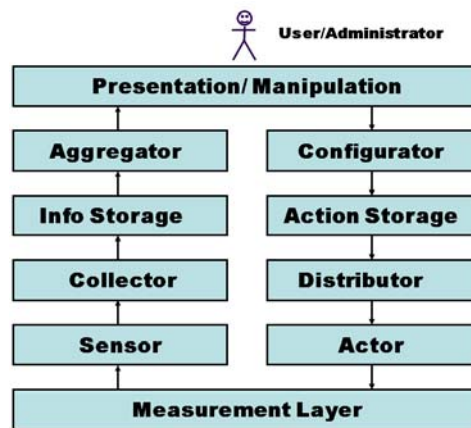


Fig. 12. Cockpit layer architecture.

The remainder of this section discusses possible realization approaches for the individual layers.

Presentation Layer: Using a large screen or a group of screens for the visualization of several information flows in parallel intuitively establishes a software cockpit. A developer, for example, could use one screen for his integrated development environment (IDE), a second one that displays the syntax of the programming language or API he is using and a third one for a web tutorial. An approach that relates to the idea of cockpits is reflected by so called desktop managers, which are supported by various operating systems (OS) and try to organize several OS desktops at the same time. Spontaneous cockpits emerge when users on their own try to organize several application windows on a desktop in order to visualize multiple parallel information flows. Those spontaneous cockpits are supported by so called “Desktop Application Integration APIs”. “Open” or “configurable” cockpits represent a framework for the integration of applications into a sub frame. “Closed” cockpits on the other hand can only be modified or extended by the vendor that distributes the cockpit. Since open cockpits are rather rare, “frame applications” are used to host the visualization of information (e.g. web browser, application frame).

Web browsers have the advantage of being platform independent and “permeable” for the most firewalls. Web browsers furthermore support the visualization of information through complex applets, portlets, Ajax scriptlets, GUI widgets or mash-ups. If a decision is made in favor of one of the above alternatives, certain infrastructural components in deeper layers are already set (e.g. web server, JSP/Java container, CMS, portal server, map server). A cockpit application, however, can also be platform neutral (e.g. the Eclipse framework). In this case, sub GUIs create and manage the parallel visualization of information of sub or plug-in applications.

For the concept of cockpits, it is especially important to be able to visualize process steps or -depending on the situation- relevant parallel information. The Eclipse framework, for instance, offers in accordance with the current development phase a set of views or perspectives on the product. Newer trends that support the openness of cockpits aim at enhancing workflows with composable representation groups, communicating applications (see OLE compounds) through story generation, chart template generators, desktop integration as well as portlet and mash-up configuration.

Aggregation and Configuration Layer: The realization of aggregators encompasses a spectrum of shared memory based direct memory aggregations. Thereby, a collector loads data into a shared memory while an aggregator reads this data and passes it on for processing. In case of applications that heavily rely on performance, one could also utilize the capabilities of modern graphics hardware and move the shared memory to the graphics card (à la GPGPU¹).

The visualization can be performed through independent components or even a modular multi MVC² concept that provides functionality for the selection of models as well as template based chart views. The configuration units capture manual interaction from the user and if applicable trigger a remote reaction or invoke a parameterized action.

Information and Action Storage Layer: In order to guarantee the functional and chronological decoupling of the huge number of parallel information acquisition and presentation activities, using modern forms of information storage (mostly relational databases) is indispensable. Not only do action storages organize and expose flexible and exchangeable workflows and action plans (or programmable controllers), but they also capture and maintain revision and history data (similar to a black box as used in aeronautics).

Collection and Distribution Layer: Collectors are programmable units that are allocated to support certain presentation modes. While they operate asynchronously in passive mode, they function as event listeners, which refresh the model of the MVC architecture and hence generate an update notification for the presenter (Controller). In active mode, depending on the acquisition strategy they query a group of distributed objects at a certain frequency and process or filter the results for the model.

Configuration instructions that were provided by the interactive elements of the presenters resemble single or compact instructions of a workflow. Distributors accept those instructions and forward them to actors or directly extract the contained information based on a predefined strategy (once or multiple times) from the storage. In addition to that, they handle the correct distribution of the instructions as well as success/failure reports of their execution.

¹ General Purpose Graphics Processing Unit

² Model-View-Controller

Transportation Layer: For the realization of the transportation layer, one could imagine IP, Industry IP or datagram based connection techniques. In local systems, shared memory or message queue techniques are often sufficient. For a cockpit that is used for the management of computer networks, standard stacks and protocols (CMI, SMI or SNMP2) as well as MIB³ databases including agents and user specific extensions are often employed.

Own solutions could also consider unicast or message queue (MQ) systems. If sockets are used, communication becomes more difficult to organize and needs to be implemented independently. MQ systems represent a central point that additionally requires a message router (similar to an integration server in J2EE systems and EAI integration). Another flexible solution that offers a little more comfort is given by so called multi agent systems (MAS) (according to the FIPA⁴ standard).

Sensor and Actor Layer: In technical processes or systems, source and destination of information are both unified in signal transmitters and telecontrols (including driver software). Computer networks often consist of firmware components (programs that monitor the status and allow for manipulation) with interfaces that ease their integration into cockpit systems. Independent stand-alone software that exposes an own GUI is only integratable via tunneling (SSH, VPN) combined with the allocation of presentation frames.

Numerous administration tools or open source solutions are not specifically designed to fit into cockpit architectures; their outputs need to be processed in an intermediate step by, for example, employing a parser with morpheme control.

The top most layers implement a Java based multi MVC concept in combination with a selection of Jfree charts. The controller handles the mapping of devices, layouts and data sets with the aid of a MySQL database, which in the same instance decouples frontend and backend. The backend system consists of the MySQL connection software, which via a gateway agent corresponds with a JADE based multi agent system.

The agents are organized into management, parse and execution agents and according to the specialization of the cockpit use open source security and management software as the sensor and actor layer.

The whole system operates in parallel tasks for the frontend, the database, the gateway agent and the single platforms. Not only does this approach make the system more stable, but it also ensures that the overall architecture is scalable. Future work aims at experimenting with mobile agents, but final results are not yet available.

3.3 First Implementation Prototype

By using the JADE framework, a first implementation for the measurement value processing and presentation could be realized as a prototype that is capable of visualizing a list of measurement readings. Using the example of capturing a ping statistic, we applied Magdeburg's cockpit prototype to a simple network security

³ Management Information Base

⁴ Foundation for Intelligent Physical Agents

scenario. The prototype's backend consists of a set of JADE containers that are distributed amongst several host systems. Those hosts represent the measurement objects. On each host we installed a different subset of available open source security programs. Those programs represent our measurement sensors. Since only a certain number of hosts can be active at a time, a configuration management agent periodically determines the current configuration and stores this data (in a configurable format) in a database.

Figure 13 shows how the first version of our cockpit prototype features an interactive way of initialization by utilizing a so called "chooser component". From a list the user can select the measurement object (host) and the sensor (program), whereby if this combination exists the visualization of the view is done based on values that are stored in the database.

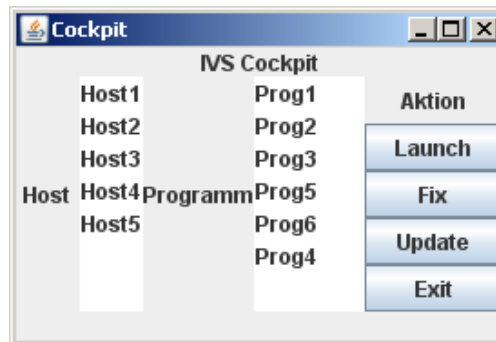


Fig. 13. Interactive cockpit initialization.

At the same time, a repository of requirements is populated by a so called "requestor component". Subsequently, a gateway agent transforms the requirements into a FIPA-ACL message (see Figure 14).

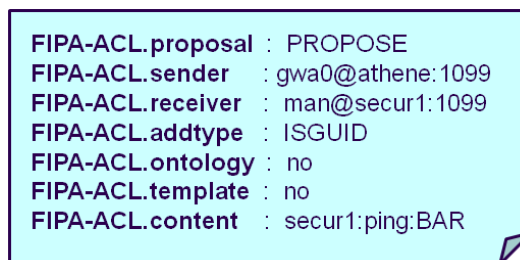


Fig. 14. A FIPA-ACL message.

Within the agent system, the message is accepted by the management agent of the corresponding JADE container (host). An executor agent is initiated which starts running a sensor program and queries the sensor program for its measurement data. Thereby, the executor agent receives commands and options from the management agent. For the ping statistics, we are using the CLI (Command Line Interface). As the

data that was obtained from the sensors is raw data (see Figure 15), in most cases it cannot immediately be used for graphical output.

```
Ping athene.cs.uni-magdeburg.de [141.44.25.20] with 32 bytes of data:
Reply from 141.44.25.20: bytes=32 time<23ms TTL=128
Reply from 141.44.25.20: bytes=32 time<5ms TTL=128
Reply from 141.44.25.20: bytes=32 time<10ms TTL=128
Reply from 141.44.25.20: bytes=32 time<8ms TTL=128

Ping statistics for 141.44.25.20:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 5ms, Maximum = 23ms, Average = 11.5ms
```

Fig. 15. Raw data of the ping statistics.

Therefore, the executor agent forwards the raw data to a management agent from where it is again forwarded to a parser agent. The parser agent on the other hand is morpheme controlled, which means that the correct morpheme file is loaded depending on the parameters that were earlier received from the management agent. As a result of the parsing operation, a key-value list is generated and passed back to the management agent (see Figure 16). From there, the data goes back to the gateway agent via FIPA-ACL where it is stored in a container.

```
program : Ping
from : [141.44.25.20]
to : [141.44.32.110]
attempt=1 : time<23ms
attempt=2 : time<5ms
attempt=3 : time<10ms
attempt=4 : time<8ms
```

Fig. 16. Key-value list generated by a parser.

An “updater component” updates the database and concurrently triggers a notification in the chart controller of the corresponding view to refresh its content. The final visualization is depicted in Figure 17.

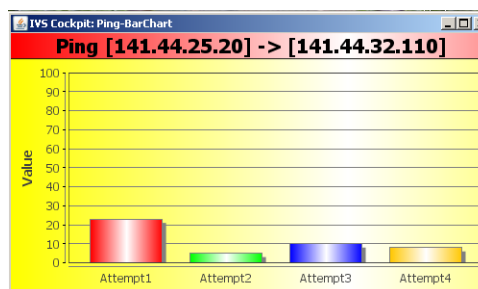


Fig. 17. Visualization of the ping statistic.

In addition to the features our current cockpit prototype exposes, we plan the following modifications and extensions:

- Substitution of the agent group with mobile agents
- Development of a configuration and management interface
- Development of adaptive compound frames for the individualization of interaction
- Parallelization of the frontend

4 Conclusion

Throughout this text, we have introduced a proposal for a measurement cockpit architecture that is adaptive to different roles, process models and system architectures. Thereby, one of our goals was to keep the complexity of our proposal as simple as possible. By following a framework approach, cockpit architectures could iteratively become adaptive to various environments. If standards and recommendations are missing, measurement instruments and visualization should be configurable to fit personal preferences. Future work on our cockpit prototype will aim at enhancing usability and performance.

Bibliography

1. Da Silva et al.: Designing Software Cockpits for Coordinating Distributed SoftwareDevelopment. Proc. of the SOFTPIT 2007, Munich, August 2007, p. 14-19
2. Dumke et al.: An Agent-Based Measurement Infrastructure. In: Abran et al.: Innovation in Software Measurement, Shaker-Verlag 2005
3. Ebert, C.; Dumke, R.: Software Measurement. Springer Verlag, 2007
4. Hansen, A.: Konzeption und prototypische Realisierung eines Metriken-Cockpits für das Software-Projektmanagement. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2008
5. Kunz et al.: Ontology-Based Design of Architectures. INMIC 2008, pp. 339-344

Further Reading

1. Bellifemine, F.; Caite, G; Greenwood, D.: Developing mulit-agent systems with JADE. John Wiley & Sons, 2007
2. Bundschuh, M.; Dekkers, C.: The IT Measurement Compedium. Springer Publ., 2007
3. Dumke, R.: Software Measurement Frameworks. Proceedings of the 3rd World Congress for Software Quality (Vol. III), Munich, September 2005, pp. 75-84
4. Laird, L. M.; Brennan, M. C.: Software Measurement and Estimation – A Practical Approach. IEEE Computer Science, 2006
5. Neumann, R.: Reengineering Deprecated Component Frameworks – A Case Study of the

- Microsoft Foundation Classes. In: Hansen et al.: Business Services, Vienna, 2009
6. Neumann, R.; Zbrog, F.; Dumke, R.: Cockpit-basierte Management-Systeme. Softwaretechnik-Trends, 29 (2009) 2, pp. 42-47
 7. Pandian, C. R.: Software Metrics – A Guide to Planning, Analysis, and Application. CRC Press Company, 2004