# System Monitoring using Constraint Checking as part of Model Based System Management

Christian Hein, Tom Ritter, Michael Wagner

Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31, Berlin 10589
{hein|ritter|wagner}@fokus.fraunhofer.de

**Abstract**: Nowadays the result of a software development process is in many cases a complex system. Critical factors that contribute to a higher degree of complexity are the size, dynamicity or heterogeneity of the developed system. Furthermore many stakeholders with different viewpoints and different interests have a need for different management features. Thus it is possible to make use of model-driven engineering techniques, methods and tools also in case of system management. We briefly described an example scenario based on CORBA Components. In this scenario we have applied constraint checking based on OCL as part of model based system management to identify malfunction of the system by automatic and semi-automatic monitoring.

## Introduction

Nowadays the result of a software development process is in many cases a complex system. Critical factors that contribute to a higher degree of complexity are the size, dynamicity or heterogeneity of the developed system. Therefore it is particularly difficult to manage such systems. Furthermore, many stakeholders with different viewpoints and different interests have a necessity for different management features. For instance a *technical* administrator is interested in the correct working of all technical resources, like used server nodes or client nodes. A *system* administrator is interested in observation of system specific policies. This may be for example security policies. The information basis could be the same for both viewpoints. However the information needs to be presented differently.

Furthermore the software development paradigm itself is changing apparently slightly from code centric to model centric development, which means the artifacts produced or used in the development lifecycle are represented as models instead of their own individual format. In other words, everything is a model. Requirements are models, designs are models, implementations are models and even the description of the system at run-time could be a model as well. Thus it is possible to make use of model-driven engineering techniques, methods and tools also in case of system management.

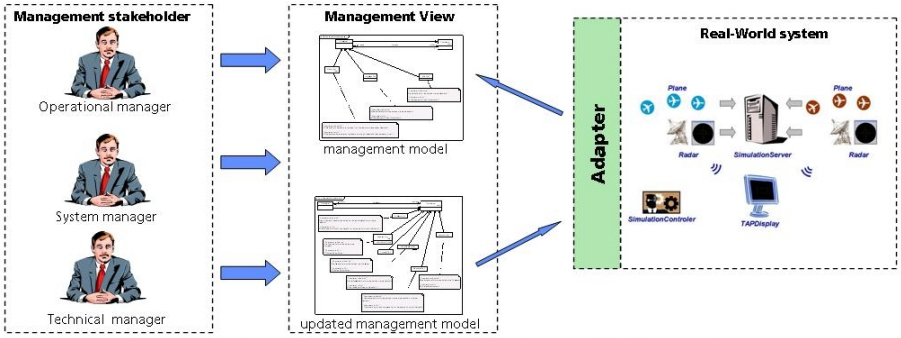


Figure 1. Model Based System Management Overview

The essential idea behind Model Based System Management (MDSM) is the model representation of a running system by using tools and technologies which are normally used for model driven system development (see Figure 1). In fact, this means that a model (e.g. notated using UML [1]) can be used not only to design a system but also to describe a running system instance. This is in particular interesting in dynamic or autonomous systems, where a great fluctuation in the internal system structure can occur (e.g. creation of instances of sub-system) and where usually the direct control over the system and its subsystems is limited. One benefit of this approach is to make use of already existing technology which is working only on models and is completely independent of the concrete system and the used platform.

Constraint checking as a part of model-based verification is one example of these existing and established technologies. It can be understood as analyzing models against expected properties. In context of model based system management constraint checking can be used to identify malfunction of the system at run-time. For example one constraint could specify that the distribution of components to different nodes should be homogenous. Another constraint could deal with security or safety aspects, for instance the definition of boundaries (maximum number of interconnections or maximum number of specific components). These constraints will be checked and evaluated during the run-time of the system. This approach enables monitoring and configuring features in case of model based system management.

In this position paper, a brief description of a CORBA Components [2] based example scenario is given. In this scenario constraint checking based on OCL [3][7] is applied as part of model based system management to identify malfunctions of the system. This can be realized using automatic and semi-automatic monitoring.

## Constraint-Checking at Run-time for Monitoring

Starting point for this approach is gathering information about the current state of the system at run-time. These data can be collected and represented as instance of a meta-model. Thus it is possible to apply model-driven techniques like transforming or querying to manipulate system by changing their models (see Figure 1). However,

this paper focuses on checking of global constraints on deployed and running systems in order to facilitate the monitoring of such systems.

Figure 2 depicts the conceptual overview of the run-time constraint checking architecture, which could be applied to various different target platforms. It is similar to Rainbow Architecture [11] or MAPE-K [12]. Model based system management in general is a mechanism, which collects run-time information and represents them in a model, which conforms to the respective meta-model. An important factor for successful use of model driven system management is the availability of an introspection mechanism in the respective platform the system is realized upon. The CORBA Component platform [2] supports introspection mechanisms. But also programming languages like Java (which also may constitute a platform) offer such capabilities [4][5]..

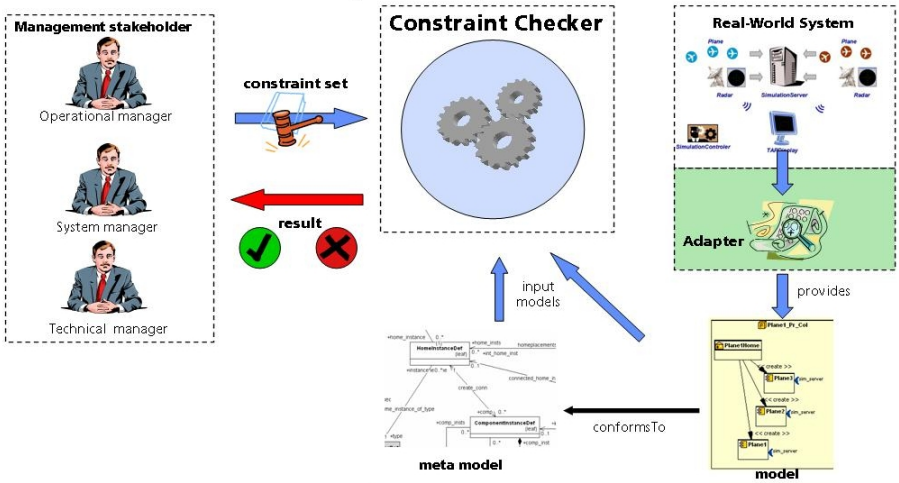


Figure 2: Concept overview

In our architecture an Adapter is responsible for using platform specific technology and introspection mechanisms to gather run-time information and to create a model out of it. Hence this component must undertake the task of doing data retrieval at first and formalization of this data secondly. The formalization requires the existence of an adequate model in which the run-time information can be represented. For this purpose the run-time adapter must have the knowledge about the corresponding meta-model.

Due to the dynamicity of the systems and depending on the platform the system is running on, it can not be assured that the Adapter collects a consistent snapshot. Perhaps a snapshot contains links between instances which are deleted during the creation of a snapshot. This problem is hard to solve. Only in cases where a global control over the system is possible, which for instance allows to freeze a system, a consistent snapshot of the system can be assured. But to freeze a whole system could also imply problems regarding the usability.

But the model validation technology presented in this paper can be used to at least identify and monitor inconsistencies in the snapshots. Depending on the importance

of such inconsistencies the quality of a snapshot can be assessed and can be taken into account when the system properties are evaluated later on. Guidelines and rules for identifying such defects and for appropriate reflection of this information will be addressed in future research. In this case, a promising area could be the extension of the corresponding meta-model with annotations which includes Fuzzy semantics properties.

After creation of a run-time model is finished the management policies can be evaluated. The inputs for this task are the run-time model provided by the Adapter, the meta-model as well as a set of constraints. The constraints must be defined by a domain expert, which has knowledge about the system and its properties. In fact he has to tell acceptable from not acceptable system properties.

In our approach the constraints are expressed at meta-model level using OCL, but it is also conceivable to use other formal language like Z as well. In contrast to traditional formal languages OCL is applicable for a large number of users and is not only applied in academic world. During the small learning curve it is also interesting for business- and system modelers, programmers or management. The advantage over program code is the great flexibility. OCL expressions can be easily changed and adapted. Furthermore these expressions are on a more abstract level than written program code.

The evaluation of the constraint set against the model is made by a constraint checker. The checker verifies each constraint in the set and provides a result. This result might be a Boolean value, the constraint is fulfilled or not. The result can also be of other types. In fact the OCL expressions could also be metrics or queries which provide other results. Therefore many possibilities exist for the subsequent treatment of the evaluation results. An alternative way is the communication of the results to the running system, thus it has the ability to react accordingly. A further option is the illustration of the evaluation result in a graphical user interface (GUI) for a system manager which is than able to analyze the resulting data. In addition after interpretation of the data the manager can formulate new OCL queries or constraints to qualify the behavior and characteristics of the run-time system.

## Application on CORBA-Components

An air-traffic management simulation based on CORBA Components is the example scenario for the proof of the described approach. Figure 3 illustrates this scenario. The air-traffic simulation consists of the five components *Plane*, *Radar*, *SimulationServer*, *TAPDisplay* and *SimulationController*. A Plane can be tracked by a Radar station and each Radar station has its area of observation. The SimulationController creates new Planes at random as well as new Radar instances. The component SimulationServer retrieves the position of all planes and finally a TAPDisplay presents the information from all radar stations.

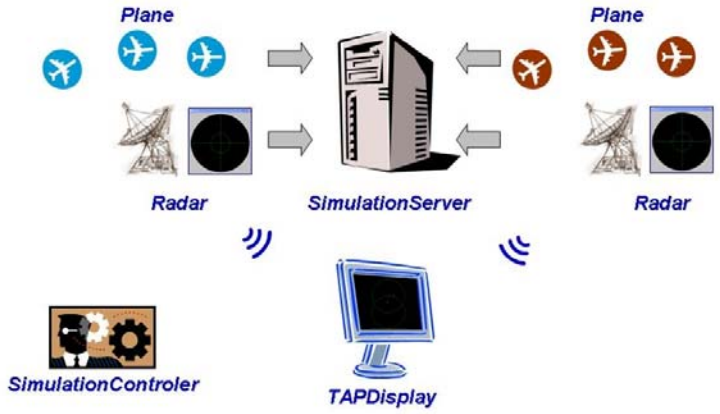


Figure 3. Sample Scenario

The scenario is a CCM based system, because CCM provides some promising features that support introspection capabilities, therefore it can be used to collect information of the system at run-time. This allows easily the creation of a snapshot of a running CCM based system with all component instances and all connections between them. The Adapter from the general architecture is implemented as a CORBA Server, which periodically creates the system snapshots and exports CCM models to the constraint checker.

The following example OCL constraint is a formal representation of the rule; that the number of planes running on one computing device shall not be significantly different from the number of planes on the other computing devices. Nevertheless it is possible that the model of the running system is inconsistent. Therefore weak constraints have to be formulated, considering these inconsistencies. The following constraint describes that a significant difference is given if the range between maximum and minimum occurring number of planes on the different devices is greater than 3. In nowadays systems usually only the maximum difference number can be changed if it is realized as a configuration value. The presented approach allows the modification of the whole constraint by the corresponding manager.

```
  context HomeInstantiation def:
  HelpSet: OrderedSet(Integer) =
HomeInstanstiation.allInstances()->
  Iterate(
  i: HomeInstantiation ;
  sum: Sequence(Integer)=Sequence{} | sum->
append(i.comp.size())
  ).asOrderedSet()

  context HomeInstantiation def:
```

```
Range: Integer =
HelpSet.last()-HelpSet.first()

context HomeDef inv:
if (identifier='PlaneHome')
then homeImpl->forAll(h|
h.instance->forAll(Range<4)
) else true endif
```

The constraint checker is realized by using the Open Source Library for OCL (OSLO) [8]. This library is an OCL2 implementation containing parser and evaluator and which can be used for arbitrary meta-models.

In order to estimate the effort need for system management when using our approach we make some performance analysis. We have done the analysis on average computer hardware connected via 100MBit Ethernet. Computers have Intel Pentium processors at clock speed of 2GHz. At first we measured the effort creating the snapshot. The adapter which is creating the snapshot is working sequentially. It traverses all component servers one by one and within the component server each container one by one and after this it examines all component instances. Another approach would be to make the snapshot in a parallel way, which means to query all component servers at the same time. The Figure 4 shows the measuring results for creating the snapshots in our CCM environment.

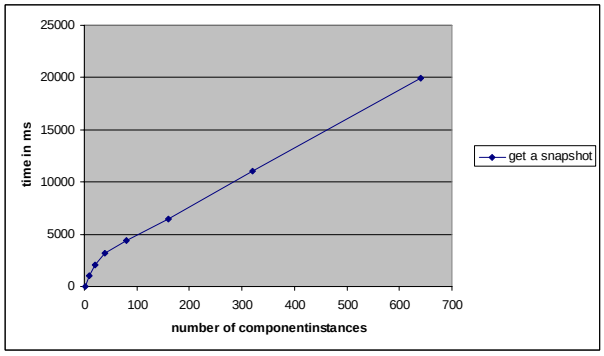


Figure 4. Measuring results of introspection mechanism

Secondly, we analyzed the effort for checking constraints against models. These models are delivered by the adapter creating the snapshots. We have executed two test cases. The first one contains only one constraint and the second one uses 10 constraints. Figure 5 depicts these measuring results. We can say that the effort of creating the snapshots and for checking constraints against models is linear.

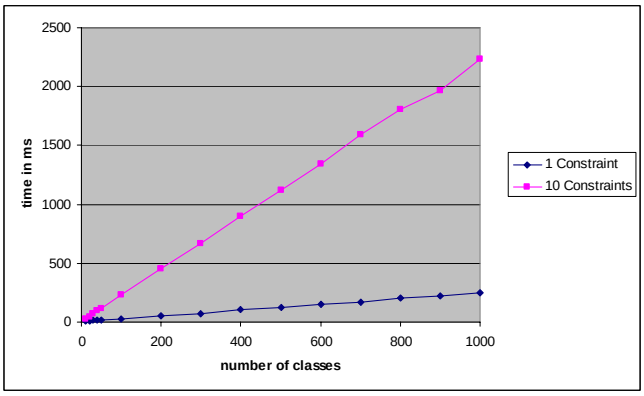


Figure 5. Measuring results of constraint checker

## Related Work

The idea of model-based system management is discussed increasingly in academic and industrial world. For example the European project ModelPlex [9] deals with this topic. It has originated a MBSM state of the art document [10]. Further work regarding monitoring of a running system has been done by Garlan [11], IBM [12] or Oracle [13]. Another interesting commercial product is the Auto Immune System of Fujitsu-Siemens [14].

## Conclusion

This paper briefly summarized an approach for constraint checking at run-time. The aim is to monitor and to verify the system at run-time as part of model based system management by using already existing model-driven technologies. The example has shown that even inconsistent models can be used for constraint checking, if the possible inconsistent parts of the models are considered as possible inconsistent. An open question is how to deal with inconsistent models. In our case study we soften the constraints to cope with inconsistent models.

There are multiple ways for reducing the inconsistency of the run-time models. At first the creation of a snapshot could be handled in different ways, depending on the size and dynamicity of the system. Alternatively to our implemented approach, it could be possible to get a complete snapshot only right before starting the system and inform the adapter only about changes within the running system. This could be useful for big and less dynamic systems. Another way to improve the consistency of the snapshot is to reduce the information which needs to be gathered by the adapter. Maybe not all aspects of the running system need to be represented in the run-time model to apply management policies. Which means the adapter needs to collect less data, which takes less time and which potentially reduces the degree of inconsistency.

However, there will always be a certain degree of inconsistency in the run-time models in particular for distributed systems, unless a system can be frozen for taking the snapshot. To deal with this problem we are currently investigating to enrich the Object Constraint Language with Fuzzy semantics.

In our approach we make use of already existing tools of the development process to derive the information for the user and his domain. This means for example to take the design tools also for system management. Future practical experiences need to show the feasibility of this approach. A particular question is how to deal with a run-time model in a design tool, without having information regarding the corresponding design process and design model. This could be the focus of future work. Another important area of future work would be the integration of a transformation tool in order to manipulate the model of the run-time with respect to re-configuration of the running system.

## Acknowledgement

The presented work in this paper is conducted in context of the MODELPLEX project. MODELPLEX is a project co-funded by the European Commission under the “Information Society Technologies" Sixth Framework Programme (02- 06). Information included in this document reflects only the authors' views. The European Community is not liable for any use that may be made of the information contained herein.

## References

[1] Object Management Group. Unified Modelling Language (UML) Specification: Infrastructure. http://www.omg.org/docs/ptc/04-10-14.pdf

[2] Object Management Group. CORBA Component Model Specification. http://www.omg.org/docs/formal/06-04-01.pdf

[3] Object Management Group. Object Constraint Language – Specification Version 2.0. http://www.omg.org/docs/ptc/05-06-06.pdf

[4] Havelund, K and Rosu G.: Monitoring Java programs with JavaPathExplorer. In Proceedings of the Workshop on Runtime Verification, volume 55 of Electmnic Notes in Theoretical Computer Science. Elsevier Publishing, 2001

[5] SUN - Java Management Extensions (JMX) Technology. http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[6] ORACLE - Oracle Business Activity Monitoring - http://www.oracle.com/technology/products/integration/bam/index.html

[7] Kleppe, Anneke & Warmer, Jos - The Object Constraint Language Second Edition, Getting Your Models Ready for MDA, 2003, Addison-Wesley

[8] OSLO - Open Source Library for OCL. Open Source project hosted at http://oslo-project.berlios.de

[9] ModelPlex Project. http://www.modelplex.org

[10] ModelPlex Deliverable. Model based System Management – State of the Art.

Deliverable D5.1a
[11] Garlan, D., Cheng, W.., Huang, A., Schmerl, B., Steenkiste, P. – Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure
[12] IBM White Paper. An architectural blueprint for autonomic computing. http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf
[13] Oracle Business Activity Monitoring - http://www.oracle.com/technology/products/integration/bam/index.html
[14] Fujitsu Siemens Auto Immune Systems - http://www.fujitsu-siemens.com/services/managed_services/autoimmune.html