March 5, 2007

RT0725

Computer Science 8 pages

Research Report

A Tool Framework for KPI Application Development

Mari Abe, Jun-Jang Jeng and Yinggang Li

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

A Tool Framework for KPI Application Development

Mari Abe
IBM Tokyo Research Laboratory
1623-14 Shimo-tsuruma, Yamato
Kanagawa 242-8502, Japan
maria@jp.ibm.com

Jun-Jang Jeng
IBM T.J. Watson Research Center
Yorktown Heights,
New York 10598, United States
jjjeng@us.ibm.com

Yinggang Li
Computer Science Department
Indiana University
Bloomington, Indiana 47404, United States
yinli@cs.indiana.edu

Abstract

This paper describes a KPI modeling environment, coined as Mozart, where modelers can use formal models to explicitly define the services of KPI and their relationships which are depicted by KPI net. Mozart provides us with methods for mining and modeling KPIs and supports smooth model transformation for generating monitoring applications based on a model driven approach. It also provide us with methods for service composition for KPI applications. We showed how it works with an example scenario for automobile data and found that "mpg" is most strongly influenced by four KPIs. The result showed which KPI should be focused for human monitoring, and it can be an initial model of monitoring applications.

1. Introduction

Managing an SOA is becoming a critical task for an enterprise. Multiple factors need to be taken into consideration: underlying physical resources, applications, services. There is an increasing focus on the measurement of business value, which requires the toolkit for SOA management. Traditional IT resources tend to be more monolithic compared with SOA services that actually manifest themselves at different levels of abstraction in a more dynamic manner. Most systems management tools are mainly oriented toward middleware, network, operating system and hardware resources. On the other hand, services are application layer components and impose different management requirement than traditional systems management.

Moreover, a service can participate in multiple business processes. The service level agreements associated with services vary according to the business process in which it participates. For example, variations of a service may be bound to different service level agreements; two services may provide the same function, but one may be optimized for ”gold” customers as opposed to those at silver- or bronze-levels. Therefore, the way to measure the values of services is also different than normal systems management. Multiple levels of metrics need to be designed and attached to appropriate levels of service stack: business processes, services, transaction performance, infrastructure health and security. For example, at the process level, an SOA management tool will provide the functions such as (1) designing key performance indicators for target business processes; (2) deploying models into SOA runtime; (3) monitoring the execution of individual processes and overall behavior of a set of processes; (4) ensuring the performance is compliant with committed service quality based on key performance indicators; (5) adapting services behavior if service quality deviates from intended range; (6) feeding information back into process models for continuous process improvement.

Among the above steps, KPI modeling, deployment and monitoring are key steps to management services in the context of business processes. Both design tools and runtime are expected to be improved and go beyond current systems management counterpart. This paper is aimed to address the issues of modeling KPIs.

Unfortunately, modeling KPIs is not regarded as main elements of existing business process modeling approach. Monitoring solutions based on business processes were proposed in [3, 11] where the goal is to help designers specify process monitoring rules as assertion codes that are based on business process execution models. The monitoring system will inform users about the status of the process with KPIs. These process-based KPI modeling approaches considered KPI as a side effect of business process modeling rather than one of the core elements of business service modeling.

This paper describes a KPI modeling environment, coined as Mozart, where modelers can use formal models to explicitly define the services of KPI and their relationships which are depicted by KPI net. KPI service models can be consequently translated and mapped into service implementation modules. Model driven approach is exploited by leveraging integrated multi-level KPI models for representing the business process level KPIs, platform-independent application KPIs, and the platform-specific implementation level KPIs. The specific functionality within Mozart that is dedicated for monitoring and controlling business / IT systems has been critical factors for the success of an enterprise. Business level monitoring and control is a near real-time, model-based discipline to, proactively and reactively, optimize and adapt business operations and IT infrastructures based on dynamic performance targets. This paper article presents the approach of defining KPI nets as services, analyze KPI nets, and translating KPI nets into service implementations.

The following sections are organized as follows: Section 2 describes KPI modeling with its design cycle and requirements. Section 3 introduces a tool framework for KPI service composition and Section 4 introduces an example scenario of automobile. Finally, we will conclude our paper by Section 5.

2. KPI modeling

In this section, we will discuss design cycle and requirements of KPI modeling. These are some of the basic requirements for modeling environments for KPI service composition. KPI design cycle and requirements of KPI modeling is important for continuous improvement of enterprises.

2.1. KPI design cycle

Process mining is a technique for exploring process flows in running applications. As many researchers have reported, the benefits of process mining are: 1. Mining processes shows the actual situation of the business process that can be used for analytic purposes. 2. Users do not need to design a process from scratch, but can use rough process descriptions [2, 15, 16]. The same benefits can be reaped from the KPI design cycle, as shown in Figure 1.

Mining KPIs and their correlations: KPIs are mined from a knowledge database with historical data. The initial monitor-enabled KPI is given by the knowledge database schema. Examples of mining KPI and their correlations were proposed in the area of data mining, where learning algorithms identify KPIs (or just variables) that are most significantly correlated with other variables [8].

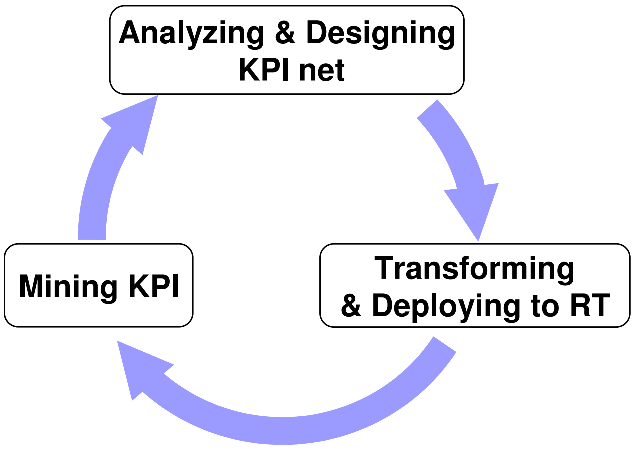


Figure 1. KPI design cycle.

Analyze and designing a KPI net: The mining step usually counts a large number of KPI correlations and does not identify which KPI should be focused for human monitoring. In this step, the number of monitor-enabled KPIs is reduced via impact analysis, sensitivity analysis, and other business-oriented analytic methods. We call the output of the analytics a KPI network or a "KPI net" for short, that describes KPI elements and the relations between them. This step also includes comparing the monitor-enabled KPIs and the to-be-monitored KPIs.

Transform and deploy KPI net to runtime: Even if an enterprise decides that the organization structure changes for reasons such as outsourcing, it might use the same KPIs to measure business performance. How the business process works and what kinds of measurement should be used are different. Therefore KPI should be designed to be insulated from the change of process model and be designed independently on process modeling at first, then models of the KPIs and of the processes can meet in the middle of the deployment or model transformation process.

2.2. KPI modeling requirements

To provide KPI design cycle support, the modeling requirements of KPIs should be defined. We believe that the following requirements should be satisfied for KPI modeling.

Why is a KPI monitored?: A KPI elements should be linked to the business goals of an enterprise. The Goal-Question-Metric (GQM) [4, 17] approach is known as an effective approach to maintaining meaningful metrics for software measurements. The business goals linked to a KPI should be defined explicitly in a model. That goal model gives the monitoring applications a focus for KPI monitoring.

What kinds of KPI should be monitored?: The defined KPI in a goal model is a high-level KPI and usually derived from finer-grained, operational KPIs. The KPI net in the previous subsection describes KPI elements and the relations among them. There are predefined types of functions: 1. computational relations that are defined arithmetically and 2. dependency relations that are discovered as correlations by mining engines. A set of KPIs might be defined during consultations with business managers. A KPI net provides the structure for KPIs and thus allows drilling down from a high-level KPI to low-level ones.

How should a KPI be calculated?: A KPI is calculated from other KPIs and/or business event attributes. Business events are retrieved from business process workflow engines, log file adapters, legacy applications implemented to emit events, and other sources [10] during monitored operations. At design time, event sources are identified with adaptation elements called sensors in a KPI net. A sensors sometimes refers to a repository of event metadata, historical data of simulations, etc. based on the kinds of events that can be retrieved from an event source. Users need to design computational relations from events for KPIs, with preconditions and post conditions to evaluate relations.

What is the context of a KPI?: When a KPI is defined, the context of monitoring the KPI should be defined as well. For example, if a KPI "product sales" is defined, categories of products can be one of contexts for monitoring, because business managers want to see what kinds of products are selling well. Such business concerns should be defined at an early stage of modeling and then transformed to a runtime configuration based on the model-driven approach. Otherwise software developers will be forced to identify the source code related to the business concerns when contexts are changed.

When should a KPI be evaluated?: Relations between KPIs consist of formulation of the relation, preconditions, and postconditions to evaluate those functions. The timing of evaluation depends on the meaning of a KPI and the functions. If a KPI is "1Q Sales" then the time to evaluate it is at the end of March. But if the KPI is something like "Sales by each representative", then the timing depends on when the sales events occurred. We believe the timing of evaluation must include at least three types and their combinations: periodic (e.g. once a month), triggered when input data satisfies conditions (e.g. when data is updated), or specific times (e.g. the end of March).

Who can monitor the KPI?: In monitoring applications, users do not make decision or take actions until they see a KPI on a dashboard. For this reason, a KPI has to have an access control policy as an attribute to be transformed into security policies on a runtime platform. However it is not always satisfactory if it only supports *"who can access which KPI"* and does not consider contextual information. For example, if a user can access KPI A and it reaches a threshold, then perhaps that person should not be allowed to access KPI B in that same monitoring context. Access control mechanisms for inter-organizational workflows have been proposed to separate inter-organizational workflow security from concrete organization-level security enforcement [12]. Similarly, access control mechanisms and runtime platforms for business performance monitoring are needed apart from modeling access control for KPI to reflect dynamic demands.

These are some of the basic requirements for modeling environments for KPI applications. In the next section, we will introduce a tool framework for KPI application development and show how it works with an example scenario.

3. Mozart: Tool framework for KPI application development

In this section, we describe a tool prototype called Mozart. Mozart provides us with a means of mining and modeling KPI net, analyzing KPI net and supports smooth model transformation to KPI applications based on the model driven approach. Mozart is implemented as a set of plugins for the Eclipse platform [7]. Some of the plugins depends on Rational Software Architect (RSA) [9] to exploit UML [13] editing functions.

Mozart consists of core plugins and extension plugins. Figure 2 shows the architecture of Mozart with a control flow from a data warehouse to a monitor model. The core plugins, the upper part of the figure, provide editing, viewing, and validating functions for designing models. The extension plugins, the lower part, provide mining engines, analytic modules, monitor model generators indicating services. The extension plugins can be added with service interfaces to adapt to users' needs. Thus, Mozart can be a powerful platform to compose services and build KPI applications.

An important feature of Mozart is that the control flow can be also designed in the editor. Figure 2 describes a control flow from a data warehouse (DW) to a monitor model [19], from the left to the right in the figure, through extension plugins indicating services. To start, a set of KPIs is extracted from the data warehouse with a data mining engine. The data warehouse can be replaced with a business process depending on the application scenario. Then a KPI net consisting of a set of KPIs and their relations is calculated from the mined KPIs and the Goal model. The goal model determines the focal KPIs according to the enterprise

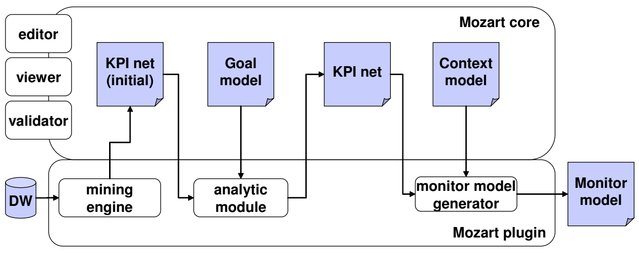
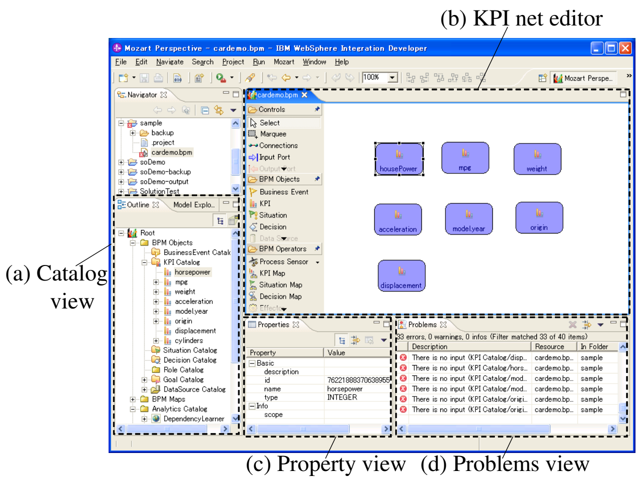


Figure 2. Mozart architecture with a control flow from data warehouse to monitor model.



Property	Value
Basic	
description	
id	76221888370638955
name	horsepower
type	INTEGER
Info	
scope	

33 errors, 0 warnings, 0 infos (Filter matched 33 of 40 items)

	Description	Resource	In Folder
⊗	There is no input (KPI Catalog/disp...	cardemo.bp...	sample
⊗	There is no input (KPI Catalog/hors...	cardemo.bp...	sample
⊗	There is no input (KPI Catalog/mod...	cardemo.bp...	sample
⊗	There is no input (KPI Catalog/mod...	cardemo.bp...	sample
⊗	There is no input (KPI Catalog/origi...	cardemo.bp...	sample
⊗	There is no input (KPI Catalog/origi...	cardemo.bp...	sample

Figure 3. Screenshot of Mozart.

strategy, so it might be replaced with a different goal model for each enterprise. Finally the KPI net is transformed to generate a monitor model by adding a context model with the generator. The context model might be customer segments for a customer relationship management application, or product categories for a supply chain management application. Such a control flow diagram can be edited in Mozart and applied as a powerful tool for developing KPI applications.

Figure 3 shows a UI screenshot of Mozart. Mozart consists of several views and editors. To exploit standard technology, the metamodel of KPI net is defined in a UML class diagram so that an instance is serialized in the XML Metadata Interchange (XMI) format [14] using the Eclipse Modeling Framework [6].

When users open the model file, elements are listed in the Catalog view, (a) in the figure. There is a set of KPI elements in the "KPI Catalog". Users can drag and drop elements into the KPI net editor (b) which displays the KPI net. The KPI net editor can be switched to other editors such as the Goal editor, Control flow editor, etc., responding to the selections of elements in the Catalog view. If the KPI elements do not have any relationships to others, the set of KPI elements are displayed as shown in the example. The element attributes are shown in the Property view, (c) in the figure. A validator validates the model before it is saved. It shows the validation results in the Problems view (d) with any validation error messages.

4. Example scenario of automobile

In this section, we describe a scenario from a mining KPI to generate a monitor model from automobile data. Once a monitor model is generated, it can be transformed and deployed on the runtime [19]. In this scenario, we explain two steps of the design cycle explained in Section 2: 1. Mining a KPI and 2. Analyzing and designing a KPI net. The step 3. Transforming and deploying to runtime is described in our previous works [1].

This scenario involves three steps: 1. Discovering KPI correlations, 2. Finding the most influential chains in a dependency graph, and 3. Saving results of the analytic tool as a KPI net for refinement. The original dataset can be downloaded from the UCI Machine Learning Repository [5]. Suppose that an automobile maker needs to check if a KPI, miles per gallons (mpg), of a new car can be improved to meet an environmental fuel efficiency objective. The question is which KPIs should be paid attention to reach such a goal in a process of assembly. The motivation is that it is difficult to design a KPI net from scratch. Also, it is difficult to predict how much each KPI influences the others.

4.1. Step1: Discovering KPI correlations

Table 1 shows sample values of automobile parameters or KPIs for each model of car. There were originally eight types of KPIs and the number of data entry is 406. From this historical data, KPI correlations were discovered by using simple linear regression, most often used for prediction between pairs of parameters.

Figure 4 is Dependency learner screenshot. It is one of the mining engine services registered in Mozart. It shows correlations between two KPIs. In this figure, it illustrates a mined correlation between cylinders and mpg with a slope -3.6, where the X-axis is the number of cylinders and the Y-axis is mpg. One cross dot on plot corresponds to one record in Table 1. The points are grouped on discrete numbers on the X-axis because the number of cylinders has discrete values. This figure shows that as the number of cylinders increases, the mpg generally decreases.

A part of WSDL [18] file of Dependency learner service is shown in Figure 5. The header, namespace declaration, definitions of types, encoding style and namespaces of soap are omitted in the figure due to space restriction.

Table 1. A part of values of automobile parameters for each type.

name	mpg	cylinders	displacement	horsepower	weight
amc-ambassador-brougham	18.0	8.0	307.0	130.0	3504.0
amc-ambassador-dpl	15.0	8.0	350.0	165.0	3693.0
amc-ambassador-sst	18.0	8.0	318.0	150.0	3436.0
amc-concord	16.0	8.0	304.0	150.0	3433.0
amc-concord	17.0	8.0	302.0	140.0	3449.0
amc-concord-d/l	15.0	8.0	429.0	198.0	4341.0
amc-concord-dl	14.0	8.0	454.0	220.0	4354.0
amc-concord-dl-6	14.0	8.0	440.0	215.0	4312.0
amc-gremlin	14.0	8.0	455.0	225.0	4425.0
amc-gremlin	15.0	8.0	390.0	190.0	3850.0

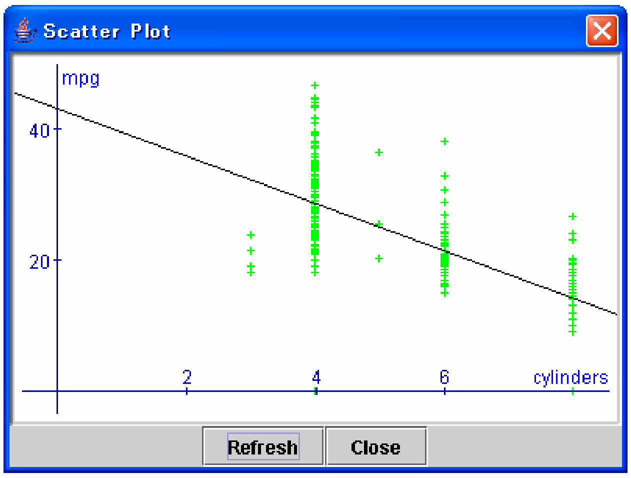


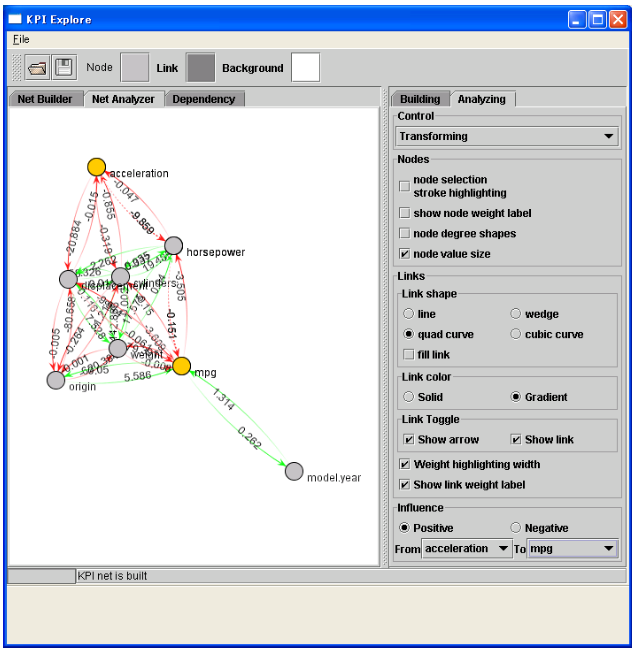
Figure 4. Dependency learner screenshot. It shows mined correlation between cylinders and mpg (slope:-3.6).

Figure 5 shows the interface of how to get a dependency and a dependency graph. In the figure, there are two operations "getDependency" and "getDependencyGraph". It also shows message formats of the two operations.

From these dependencies, you can get a dependency graph. Figure 6 is another extension tool for a service "KPI net explore". that shows 8 KPIs and their correlations in one graph. Each node indicates a KPI and each weighted, directed edges represents correlations as influence between those two nodes. The weight is the numeric influence between KPIs, which may be positive or negative. Users can analyze the graph interactively with this tool. In the next step, we will show how to analyze a dependency graph with this tool.

```
<wsdl:message name="getDependencyResponse">
   <wsdl:part name="getDependencyReturn"
              type="xsd:double"/>
</wsdl:message>
<wsdl:message name="getDependencyGraphResponse">
   <wsdl:part name="getDependencyGraphReturn"
              type="impl:DependencyGraph"/>
</wsdl:message>
<wsdl:message name="getDependencyRequest">
   <wsdl:part name="x" type="impl:ArrayOf_xsd_double"/>
   <wsdl:part name="y" type="impl:ArrayOf_xsd_double"/>
</wsdl:message>
<wsdl:message name="getDependencyGraphRequest">
   <wsdl:part name="KPIList_X" type="impl:ArrayOfKPI"/>
   <wsdl:part name="KPIList_Y" type="impl:ArrayOfKPI"/>
   <wsdl:part name="dependencyList"
              type="impl:ArrayOf_xsd_double"/>
</wsdl:message>
<wsdl:portType name="DependencyLearner">
   <wsdl:operation name="getDependency"
         parameterOrder="in0 in1">
      <wsdl:input message="impl:getDependencyRequest"
                     name="getDependencyRequest"/>
      <wsdl:output message="impl:getDependencyResponse"
                      name="getDependencyResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getDependencyGraph"
         parameterOrder="in0 in1 in2">
      <wsdl:input
               message="impl:getDependencyGraphRequest"
                  name="getDependencyGraphRequest"/>
      <wsdl:output
              message="impl:getDependencyGraphResponse"
                 name="getDependencyGraphResponse"/>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="DependencyLearnerSoapBinding"
              type="impl:DependencyLearner">
   <wsdlsoap:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="getDependency">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getDependencyRequest"/>
      <wsdl:output name="getDependencyResponse"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getDependencyGraph">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getDependencyGraphRequest"/>
      <wsdl:output name="getDependencyGraphResponse"/>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="DependencyLearnerService">
   <wsdl:port
          binding="impl:DependencyLearnerSoapBinding"
             name="DependencyLearner">
     <wsdlsoap:address location="http://localhost:8080/
                     axis/services/DependencyLearner"/>
   </wsdl:port>
</wsdl:service>
```

Figure 5. WSDL file of Dependency learner.



Control: Transforming

Nodes
- [ ] node selection stroke highlighting
- [ ] show node weight label
- [ ] node degree shapes
- [✓] node value size

Links

Link shape: line [✓ quad curve] wedge cubic curve
- [ ] fill link

Link color: Solid [✓ Gradient]

Link Toggle
- [✓] Show arrow
- [✓] Show link
- [✓] Weight highlighting width
- [✓] Show link weight label

Influence: [✓ Positive] Negative

From acceleration To mpg

Figure 6. KPI net explore shows dependency graph of automobile data with 8 KPIs and their correlations. It also shows the most influential chain.

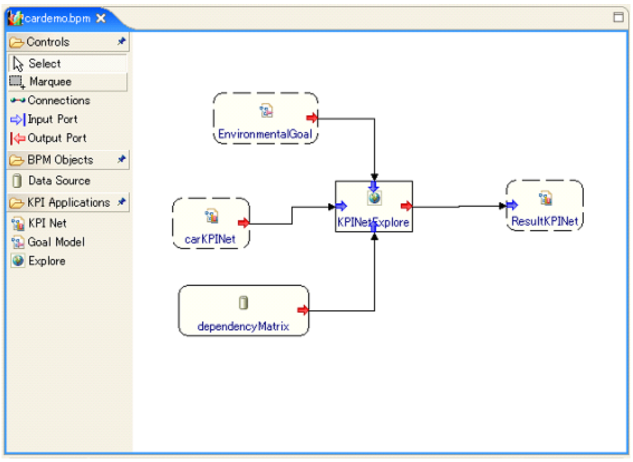


Figure 7. Control flow to find the most influential chain.

4.2. Step 2: Finding the most influential chains in a dependency graph

A goal model is needed to recognize the crucial KPIs and their relations among the other KPIs. In this scenario, “Environmental goal” is linked to mpg in goal model that gives the dependency graph a focal KPI, mpg, to start the analysis. Figure 7 shows a control flow in the Control flow editor with three inputs, which are the goal model, a set of KPIs, and the dependency graph. The last two of these were prepared in the Step 1. The element in the center of the figure shows KPI net explore, one of analytic services in Mozart. It can find the most influential chain from these three inputs and output a KPI net.

A part of WSDL file of KPI net explore is shown in Figure 8. It includes two operations that is “getInfluenceIn” and “getInfluenceOut”. “getInfluenceIn” can find KPIs which influence the focal KPI. “getInfluenceOut” can find KPIs which the focal KPI influences. These two operations take three inputs described in the control flow in Figure 7 and one threshold. The threshold is used to limit the number of KPIs to save a new KPI net explained in the next step.

The example scenario is to discover which KPIs should be paid attention to drive the goal in a process of car assembly. It will not be sufficient to monitor mpg only given other KPIs might influence to it directly or indirectly. In Figure 6, there is an functional option to find the most influential chain in the lower right. It enables impact analysis by selecting a starting KPI node and an ending node. With dependency graph in place, it then runs an algorithm to find concurrently the paths of most negative and positive influence. Selection can be obtained from a goal model as well when it is invoked. In the figure, selected nodes are highlighted and the most influential chains are shown as dotted line. As outlined in Figure 9, the most influential chain algorithm is an adaptation of the classic shortest path algorithm, where two ”influence” metrics, positive and negative, are kept at each node as distance metric, and multiplication replaces summation at each relaxation step. For instance, two consecutive negative influences may turn into a positive one. The algorithm will find both paths with most positive impact and negative impact the starting KPI may have on the ending KPI.

4.3. Step 3: Saving the results of the analytic tool as a KPI net for refinement

The analytic tool supports not only discovering the most influential chains but can also save the result as a KPI net based on a threshold. Figure 10 shows a resulting KPI net saved by the analytic tool. The threshold of absolute values for impacts was set to be > 1.0. The righthand element in the KPI net editor is an element for mpg, which is most strongly influenced by four KPIs, the origin (where the car was made), the number of cylinders, the model year, and acceleration. Users need to refine the KPI net using Mozart editing functions to input which event source is used, how the KPI is calculated from the events, and so on. But at least

```
   <wsdl:message name="getInfluenceInRequest">
      <wsdl:part name="goalModel" type="impl:GoalModel"/>
      <wsdl:part name="dependenchGraph"
                 type="impl:DependencyGraph"/>
      <wsdl:part name="kpiNet" type="impl:KPINet"/>
      <wsdl:part name="threshold" type="xsd:double"/>
   </wsdl:message>
   <wsdl:message name="getInfluenceInResponse">
      <wsdl:part name="getInfluenceInReturn"
                 type="impl:KPINet"/>
   </wsdl:message>
   <wsdl:message name="getInfluenceOutResponse">
      <wsdl:part name="getInfluenceOutReturn"
                 type="impl:KPINet"/>
   </wsdl:message>
   <wsdl:message name="getInfluenceOutRequest">
      <wsdl:part name="goalModel" type="impl:GoalModel"/>
      <wsdl:part name="dependencyGraph"
                 type="impl:DependencyGraph"/>
      <wsdl:part name="kpiNet" type="impl:KPINet"/>
      <wsdl:part name="threshold" type="xsd:double"/>
   </wsdl:message>
   <wsdl:portType name="KPINetExplore">
      <wsdl:operation name="getInfluenceIn"
            parameterOrder="in0 in1 in2 in3">
        <wsdl:input message="impl:getInfluenceInRequest"
                       name="getInfluenceInRequest"/>
        <wsdl:output message="impl:getInfluenceInResponse"
                        name="getInfluenceInResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getInfluenceOut"
            parameterOrder="in0 in1 in2 in3">
       <wsdl:input message="impl:getInfluenceOutRequest"
                      name="getInfluenceOutRequest"/>
       <wsdl:output message="impl:getInfluenceOutResponse"
                       name="getInfluenceOutResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="KPINetExploreSoapBinding"
                 type="impl:KPINetExplore">
      <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getInfluenceIn">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getInfluenceInRequest"/>
         <wsdl:output name="getInfluenceInResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getInfluenceOut">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getInfluenceOutRequest"/>
         <wsdl:output name="getInfluenceOutResponse"/>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="KPINetExploreService">
      <wsdl:port binding="impl:KPINetExploreSoapBinding"
                    name="KPINetExplore">
        <wsdlsoap:address location="http://localhost:8080/
                              axis/services/KPINetExplore"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Figure 8. WSDL file of KPI net explore.

$Root\ KPI : s$
$Each\ vertex\ v\ has\ variable\ D_{Pos}[v],\ D_{Neg}[v]\ where$
$\quad D_{Pos}[v] :\ most\ positive\ influence\ s\ has\ on\ v$
$\quad D_{Neg}[v] :\ most\ negative\ influence\ s\ has\ on\ v$
$Link\ weight : d(u,v) \neq 0$
$\quad d(u,v) :\ influence\ factor\ between\ u\ and\ v$
$Initially : D_{Pos}[s] = 1.0;\ D_{Neg}[s] = 1.0;$
$\qquad\qquad v \neq s : D_{Pos}[v] = 0.0, D_{Neg}[v] = 0.0$
$Relaxation\ step\ over\ edge(u,v) :$
$D_{Pos}[v]$
$\quad = max\{D_{Pos}[v], D_{Pos}[u] * d(u,v)\}\ if\ d(u,v) > 0$
$\quad = max\{D_{Pos}[v], D_{Neg}[u] * d(u,v)\}\ if\ d(u,v) < 0$
$D_{Neg}[v]$
$\quad = min\{D_{Neg}[v], D_{Neg}[u] * d(u,v)\}\ if\ d(u,v) > 0$
$\quad = min\{D_{Neg}[v], D_{Pos}[u] * d(u,v)\}\ if\ d(u,v) < 0$

Figure 9. Basis of most influential chain algorithm.

users do not need to design the KPI net from scratch.

5. Conclusion

In this paper, we discussed design cycle and requirements of KPI modeling. The KPI design cycle consists of three steps; 1. Mining KPIs, 2. Analyzing and designing a KPI net, and 3. Transforming and deploying it to a runtime. Our design cycle supports efficient development of KPI applications since KPIs are mined from data repository. We introduced the requirements for KPI modeling and a tool framework, Mozart, aimed at satisfying these requirements. Mozart provides us with methods for mining and modeling KPIs and supports smooth model transformation for generating monitoring applications based on a model driven approach. Since Mozart is implemented by exploiting standard technology such as the Eclipse platform, it will not enforce constraints that depends on proprietary technology.

We showed how it works with an example scenario for automobile data with three steps: 1. Discovering KPI correlations, 2. Finding the most influential chains in a dependency graph, and 3. Saving the results of the analytic tool as a KPI net for refinement. We showed “mpg” is most strongly influenced by four KPIs, the origin, the number of cylinders, the model year, and acceleration. In the future, we will apply our tool to large-scale service compositions in which a high volume of KPIs exists.

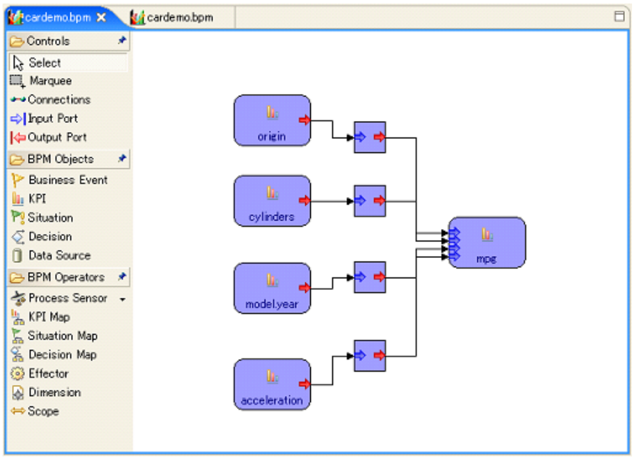


Figure 10. Result showing KPI net with KPI relations (Threshold of absolute values of impacts > 1.0).

References

[1] M. Abe, T. Koyanagi, J. Jeng, and L. An. An Environment of Modeling Business Centric Monitoring and Control Applications. In *Proceedings of IEEE International Conference on e-Business Engineering (ICEBE 2006)*, October 2006.

[2] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Advances in Database Technology (EDBT'98)*, volume 1377 of *Lecture Notes in Computer Science*, pages 467–483, 1998.

[3] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Process. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*, pages 269–282, December 2005.

[4] V. R. Basili and H. D. Rombach. The TAME project: towards improvement-oriented softwareenvironments. *IEEE Transactions of Software Engineering*, 14(6):758–773, 1998.

[5] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998.

[6] Eclipse Foundation. Eclipse Modeling Framework. `http://www.eclipse.org/emf/`.

[7] Eclipse Foundation. Eclipse Platform. `http://www.eclipse.org/`.

[8] M. Ettl, B. Zadrozny, P. Chowdhary, and N. Abe. Business Performance Management System for CRM and Sales Execution. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, 2005.

[9] IBM Corporation. IBM Rational Software Development Platform. `http://www-306.ibm.com/software/info/developer/busvalue.jsp`.

[10] IBM Corporation. Integrate event management with Common Event Infrastructure. `http://www-128.ibm.com/developerworks/library/ac-cei/`.

[11] A. Lazovik, M. Aiello, and M. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution. In *Proceedings of International Conference on Service Oriented Computing (ICSOC'04)*, pages 94–104, November 2004.

[12] J. S. P. Myong H. Kang and J. N. Froscher. Access Control Mechanisms for Inter-organizational Workflow. In *The sixth ACM symposium on Access control models and technologies*, pages 66–74. ACM Press, 2001.

[13] Object Management Group, Inc. OMG Unified Modeling Language Specification. `http://www.omg.org/technology/documents/formal/uml.htm`.

[14] Object Management Group, Inc. XML Metadata Interchange(XMI) Specification. `http://www.omg.org/docs/formal/03-05-02.pdf`.

[15] W. van der Aalst, T. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004.

[16] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454, 2005.

[17] F. van Latum, R. van Solingen, M. Oivo, B. Hoisl, D. Rombach, and G. Ruhe. Adopting GQM-Based Measurement in and Industrial Environment. *IEEE Software*, 15(1):78–86, 1998.

[18] Web Service Definition Language. `http://www.w3.org/TR/wsdl`.

[19] L. Zeng, H. Lei, M. Dikun, H. Chang, K. Bhaskaran, and J. Frank. Model-Driven Business Performance Management. In *IEEE International Conference on e-Business Engineering (ICEBE2005)*, pages 295–304, October 2005.