Performance Evaluation Lab
Dipartimento di Elettronica e Informazione
Politecnico di Milano - Italy

Java Modelling Tools

system manual

Version 0.1, June 30, 2006

Contents

Chapter 1

Introduction

This manual is focused on advanced architectural structures of JMT. It is intended for advanced users that want to interface JMT computational engines with third party software, or to interact directly with the XML data layer. More details on the computational algorithms are also provided.

This manual will not cover any aspect of normal user interaction with the suite, for such documentation please refer to *JMT users manual*.

Chapter 2

JMVA

2.1 Architecture of the tool

JMVA has been designed to be very flexible. One important feature is the complete separation between GUI and computation engine obtained through an XML layer as shown in Figure 2.1. This architecture allows reuse of the analytic engine in other projects by simply providing a suitable XML input file (see section 2.2 for details).

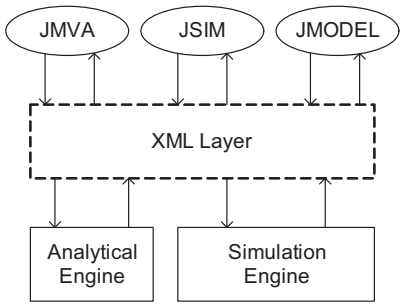

Figure 2.1: JMT Architecture

Model solution with analytical engine can be started by executing the **solve(File XMLFile)** method of the **jmt.gui.exact.link.SolverDispatcher** class. The `XMLFile` parameter is a well formed XML File according to *JMTmodel.xsd* schema described in section 2.2. At the end of the computation, performance indices will be placed into the **solutions** element of input file.

2.2 XML format

JMVA XML format is simple and can be written even by hands. The syntax of that file is specified in *JMTModel.xsd* and is graphically represented in Figure 2.2.

The root element is **model** and can include an optional **description** of the model, a section with **solutions** (that is parameterized by the solver engine) and the section **parameters** used to specify the network to be solved.

The section **parameters** contains the section **classes** that has for attribute the global **number** of classes and can include from one to infinite **openclass** or **closedclass**. **openclass** must specify a **name** and arrival **rate**, while **closedclass** must specify a **name** and **population**.

For example to define one closed class with $N = 10$ and one open class with $\lambda = 0.5$ the code will be:

```
<classes number="2">
    <closedclass name="ClosedClass" population="10"/>
    <openclass name="OpenClass" rate="0.5"/>
</classes>
```

The other section of **parameters** is **stations**. This element has **number** as attribute (like classes) and is composed from one to infinite elements of the following types:

delaystation is a delay station (infinite server)

listation is a load independent station

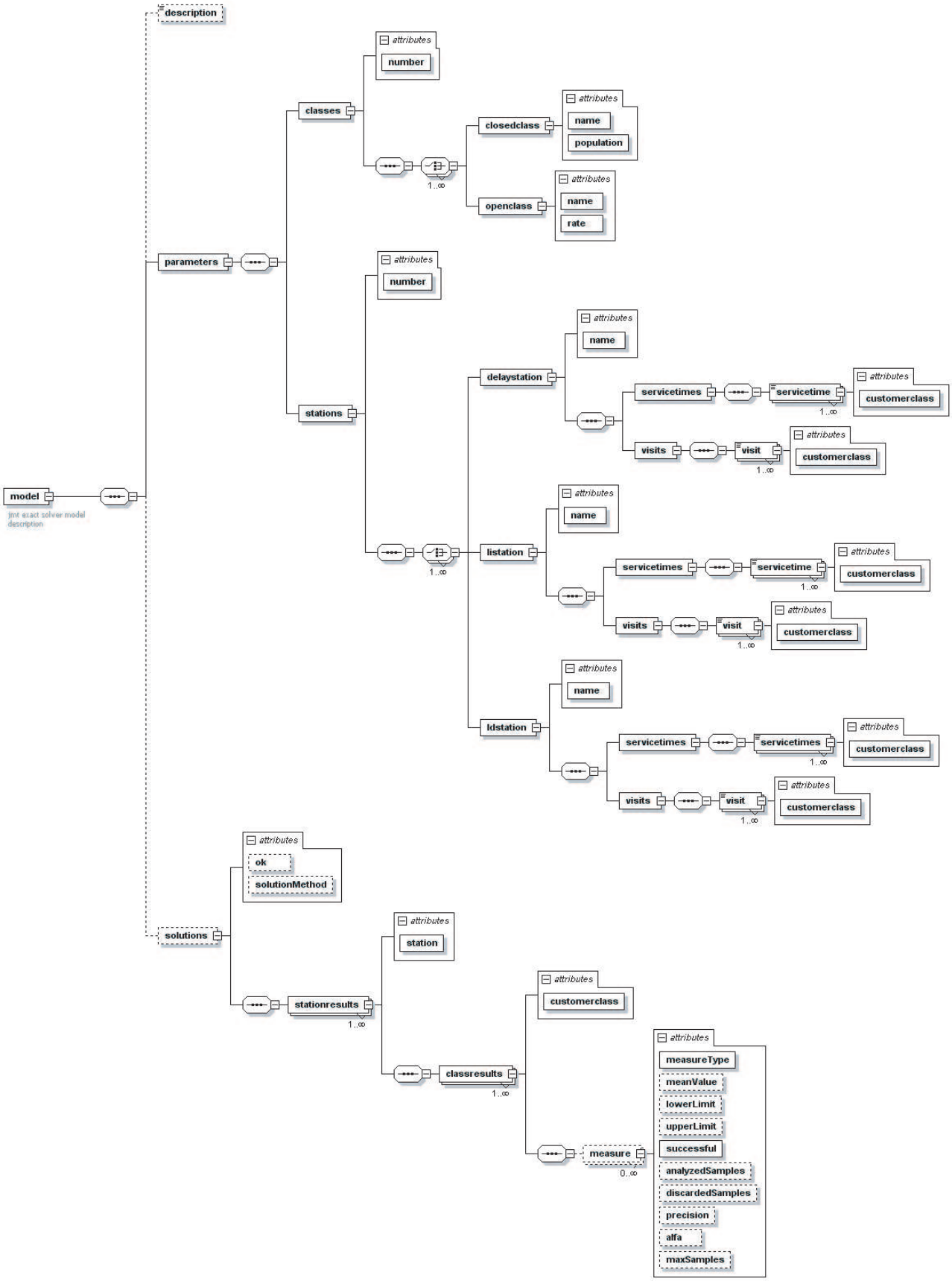

Figure 2.2: JMTModel.xsd stylesheet graphic representation

ldstation is a load dependent station

All the elements are defined with the same procedure. For each station the **name** attribute should be specified and two elements named **servicetimes** and **visits** must be included. **servicetimes** must include a list of at least one **servicetime** element and **visits** must include a list of at least one **visit** element. Both **visit** and **servicetime** elements have a `double` numeric value and each element has an attribute named **customerclass** that is used to associate a value of service time/visit with the corresponding customer class.

The only exception is **servicetime** parameter for a load dependent station. In this case the value is not a double but is a list of `double` separated by the special character **;** . The list is ordered for ascending values of number of jobs in the station, starting from 1 to number of customer N of the closed class of the model.

For example in a model with one closed class, named *Class1*, with $N = 5$ and three stations: a load independent with service demand 1, a delay with service demand 2 and a load dependent with $D(N) = N + 2$, the following code should be used for the station definition:

```
<stations number="3">
    <listation name="LoadIndependent">
        <servicetimes>
            <servicetime customerclass="Class1">
                1.0</servicetime>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </listation>
    <delaystation name="Delay">
        <servicetimes>
            <servicetime customerclass="Class1">
                2.0</servicetime>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </delaystation>
    <ldstation name="LoadDependent">
        <servicetimes>
            <servicetimes customerclass="Class1">
                3.0;4.0;5.0;6.0;7.0</servicetimes>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </ldstation>
</stations>
```

Also the computed performance indices are stored in the same XML document. **solutions** has two attributes: **ok** that indicates if the computation was successful and **solutionMethod** that indicates if solution was obtained through analytical engine or simulator. Inside a **solutions** element there is a list of one or more **stationresults**, an element with station **name** as its attribute that contains one or more **classresults**, an element with class name (**customerclass**) as its attribute. This peculiar structure is needed to store a matrix of results. Each **classresults** element contains any number of **measure** elements that are used to store computed performance indices. **measure** has the following attributes:

measureType type of performance index, can be one between *Queue length*, *Throughput*, *Residence time* and *Utilization*

successful boolean field that indicates if measure was computed correctly

meanValue (optional) computed value. This field is optional and will be always present if successful=true

lowerLimit (optional) lower limit of confidence interval. As MVA produces *exact* solution, this field is used only when model is solved with simulator

upperLimit (optional) upper limit of confidence interval. As MVA produces *exact* solution, this field is used only when model is solved with simulator

analyzedSamples (optional) number of samples analyzed by the simulator

discardedSamples (optional) number of samples discarded by the simulator

precision (optional) maximum relative error allowed by simulator

alfa (optional) confidence interval required to simulator

maxSamples (optional) maximum number of samples allowed by simulator

Note that only the first three elements of the above description will be saved by JMVA. The others are used when the model is solved using the simulator JSIM. For example, a model with one class (named *Class1*) and two stations (named *Station1* and *Station2*) will produce the following solutions:

```
<solutions ok="true" solutionMethod="analytical">
    <stationresults station="Station1">
        <classresults customerclass="Class1">
            <measure meanValue="9.930828575679609"
                measureType="Queue length"
                successful="true"/>
            <measure meanValue="5.467534818643117"
                measureType="Throughput"
                successful="true"/>
            <measure meanValue="1.8163265356477696"
                measureType="Residence time"
                successful="true"/>
            <measure meanValue="0.9999999999987986"
                measureType="Utilization"
                successful="true"/>
        </classresults>
    </stationresults>
    <stationresults station="Station2">
        <classresults customerclass="Class1">
            <measure meanValue="0.06917142432039082"
                measureType="Queue length"
                successful="true"/>
            <measure meanValue="5.467534818643117"
                measureType="Throughput"
                successful="true"/>
            <measure meanValue="0.01265130019557098"
                measureType="Residence time"
                successful="true"/>
            <measure meanValue="0.06469628980696993"
                measureType="Utilization"
                successful="true"/>
        </classresults>
    </stationresults>
</solutions>
```

It is important to point out that JMVA stores and loads model and results in *exactly* the same XML format used to dialogue with the analytical engine. The best practice to learn rapidly its syntax is to create a model using JMVA GUI, store it in any place and open it with a text editor.

Appendix A

Basic Definitions

Customer : or *job* or *transaction* is the element that will require service from our network. For example, it can be an *http request*, a *database query*, or an *ftp download command*.

Class : a group of elements that are statistically equal. The customers of a class have the same workload intensity and the same average service demands.

Station : or service center, it represents a system resource. Customers arrive at the station and then, if necessary, wait in queue, receive service from server then depart.

Workload Intensity (λ or N) : rate at which customers of a given class arrive at the system. Each class has its own workload intensity. If a class is closed the constant *number of customers* N that are present in the system must be provided, if a class is open the *arrival rate* λ of customers to the system is requested.

Population Mix (β_i) : the ratio between *number of customers* of closed class i and the total number of customers in the system: $\beta_i = N_i / \sum_k N_k$

Service Time (S_k): average time of service required at each visit to resource k; it is computed as the ratio of the busy time to the number of system completions. It is an alternatively way to describe the service requirement.

Visit (number of -) (V_k) : average number of visits that a customer makes at station k during a complete execution; it is computed as the ratio of the number of completions at resource k to the number of system completions. If resource k is a delay center representing a client station, it is a convention assign a unitary value to number of visits to this station.

Service Demand (D_k) : average service requirement of a customer, that is the total amount of service required by a complete execution at resource k. In the model it is necessary to provide separate service demand for each pair service center-class. It is given by $D_k = V_k * S_k$.

Throughput (X_k) : rate at which customers are executed by station k, that is the number of completions in a time unit.

Queue length (Q_k) : average number of customers at station k, either waiting in queue and receiving service.

Response Time (R_k): average time interval between the instant a customer arrives at station k and the instants it terminate its servicing.

Residence Time (W_k) : average time that a customer spent at station k during a complete interaction with the system. It includes time spent queueing and time spent receiving service. It does not correspond to *Response Time* R_k of a station since $W_k = R_k * V_k$.

Utilization (U_k) : proportion of time in which the station k is busy or, in the case of a delay center, is the average number of customers in the station (see Little Law [Lit61]).

System Throughput (X) : rate at which customers perform an entire interaction with the system. This is the aggregate measure of *Throughput*: $X = X_k / V_k$.

System Response Time (R): correspond to the intuitive notion of response time perceived by users, that is, the time interval between the instant of the submission of a request to a system and the instant the corresponding reply arrives completely at the user. It is the aggregate measure of *Residence Times*: $R = \sum_k W_k$.

Average number of customers in the system (N): the average number of customer in the system, either waiting in queue or receiving service. It is the aggregate measure of *Queue Length*: $N = \sum_k Q_k$.

Bibliography

[Lit61] John D. C. Little. *A Proof of the Queueing Formula L= λW. Operations Research*, 9:383–387, 1961.