JAMS – A Framework for Natural Resource Model Development and Application

S. Kralisch and P. Krause

Department of Geoinformatics, Hydrology and Modelling, Friedrich-Schiller-University, Jena, Germany
sven.kralisch@uni-jena.de

Abstract: Current challenges in sustainable management of water resources have created demand for integrated, flexible and easy to use hydrological models which are able to simulate the quantitative and qualitative aspects of the hydrological cycle with a sufficient degree of certainty. Existing models which have been developed to fit these needs are often constrained to specific scales or purposes and thus can not be easily adapted to meet different challenges. As a consequence resulting from this shortage a number of modelling frameworks have been developed, e.g. the Object Modelling System (OMS). In order to enhance the capabilities of OMS towards special demands from model developers we created the Jena Adaptable Modelling System (JAMS). Built on the basis of OMS, JAMS focuses especially on flexibility during the development of new model components and less on easy integration of existing ones. This paper will present an overview of the JAMS architecture and sketches the model component development with JAMS.

Keywords: Modular Modelling Systems; Hydrology; Integrated Water Resources Management

1 INTRODUCTION

With the implementation of the European Water Framework Directive (WFD – European Union [2000]) prognostic modelling for sustainable management of water resources has become even more important than it was before. The goals set up by the WFD require a stronger integrative and multidisciplinary approach than usually practiced in the last decades. In addition to quantitative and qualitative hydrological issues, socio-economic and legislative objectives must be considered to find the best solutions for maintenance or improvement of water quality in European water bodies.

Although such an interdisciplinary and holistic approach is the most promising way to reach the goals set up by the WFD, it also introduces new problems, which have to be solved in advance. The most obvious problem is that each discipline involved in the development of strategies for sustainable management of water resources uses their own methods and tools for prognostic simulation and modelling of the single processes of the water cycle throughout Europe. This is true not only from the multidisciplinary standpoint but also from a regional point of view. Scientists from one discipline in one part of Europe often use different models for the same purpose than other scientists in other parts of the continent because the constraints and environmental circumstances are different.

The most significant differences between the single models or modelling systems applied in Europe and worldwide are the specific model cores which simulate the single processes. On the other hand, all models or modelling systems have systematic functionalities (e.g. data in- and output) which are essentially common for all models even if they had been implemented in different ways.

For future proof model development and application, a modular approach which divides the systematic routines from the scientific parts is the most promising approach. Such an approach should provide the basic functionality for data in- and output, application and communication of the single model components as well as an application programming interface (API) for the implementation of the scientific methods in the form of encapsulated programme modules. The most relevant benefit of such a framework for model developers would be to enable them to concentrate on only the implementation of most suitable methods and always be confronted with a familiar interface and modelling environment. During the last years, various efforts have

focused on developing such systems with different results. On one side, systems for coupling whole models are widely used and succesfully applied in practice. However, only few modelling systems exist that are suited to couple components at the level of single process implementations (Leavesley et al. [1996]). Often, these systems are characterised by tight system constraints which interfere with ease of use.

Based on the Object Modeling System (OMS) [David et al., 2002; Kralisch et al., 2005] we developed the Jena Adaptable Modelling System (JAMS) which provides the mentioned features. This paper deals with its introduction, the current development and implementation of suitable programme modules.

2 JAMS Requirements

The JAMS architecture is strongly influenced by the requirements of developing hydrologic models for integrated water resources management (IWRM). The following characteristics of such models were important for the JAMS development:

- Their catchment discretisation uses different distribution concepts e.g. lumped , semi distributed or fully distributed approaches.
- Their temporal and spatial resolutions cover broad scales and often involve nested approaches.
- Their process descriptions can range from mostly physically based to empirical / conceptional.
- They need to consider topology-based processes like lateral routing.
- They need to consider a large number of different processes describing the catchments hydrology as well as solute transport dynamics. Depending on the management task at hand and available input data different alternatives of process descriptions may be suited more or less.

Based on these characteristics we defined the following requirements to be met by JAMS:

- All process implementations need to be technically independent from the spatial representation of the simulated catchment.
- JAMS must allow for an arbitrary configuration of the spatial and temporal domains a process simulation is applied for.
- JAMS must be able to integrate process descriptions ranging from simple empirical to complex physically based algorithms.

In order to meet these requirements we developed the JAMS architecture which is being sketched in the following section.

3 JAMS Architecture

In order to achieve a maximum platform independence, JAMS was implemented in JAVA. The basic JAMS concept is the representation of complex simulation models as aggregates of well-defined model components. The functionality of these model components can range from data I/O or implementations of single processes to complex sub-models. Each JAMS model is based on a model description which defines the various model components that are used. This XML-based document defines how the different model components are assembled and which data must be exchanged between the components. The model components together with the model description define a JAMS model and can be seen as the JAMS knowledge part.

The model setup, the model execution and the communication between JAMS components is established by the JAMS runtime system. The JAMS data types and basic functionalities like I/O mechanisms or unit conversion are provided by the JAMS core library. This library together with the runtime system can be regarded as JAMS core. The general layout of JAMS is shown in fig. 1.

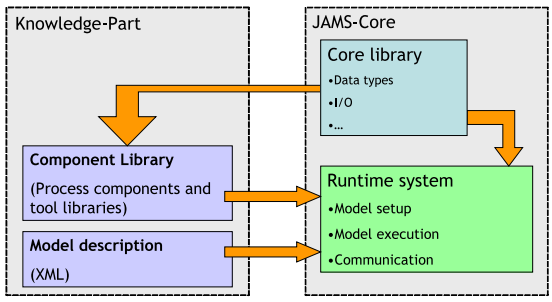

Figure 1: Common layout of JAMS

4 JAMS Models

4.1 Components

The building blocks of every JAMS model are *components*. Each of them is characterised by *(i)* a set of attributes that can be used to exchange data with other components and *(ii)* three procedures *init()*, *run()* and *cleanup()* which implement the component's functionality. These procedures are executed

by the JAMS runtime system at three according runtime stages:

1. During the *init stage* the init()-procedures of all JAMS components are executed. This stage is executed only once at the very beginning of model execution. Example functionalities executed during this stage are opening files or setting initial process parameters.

2. During the *run stage* the run()-procedures of all JAMS components are executed. This stage is executed repeatedly for different points in time and space. The functions executed during this stage actually implement the models functionality, e.g. the calculation of the soil-water-balance or evapotranspiration.

3. During the *cleanup stage* the cleanup()-procedures of all JAMS components are executed. This stage is executed only once at the very end of model execution. An example functionality executed during this stage is the closing of files.

Depending on the component's functionality not all of the three procedures need to be implemented. However, a typical JAMS component using all of them is a component reading time series data from a file. Such a component would create a file reader object in the init()-procedure. Then this reader object would be used within the run()-procedure to consecutively read data from the file. Finally, the file would be closed within the cleanup()-procedure after all data have been read.

In order to execute a component's run()-procedure repeatedly for different points in time or space, JAMS offers special *Context Components*.

4.2 Context Components

Context components work as containers for other components and control the execution of their run()-procedures. In this way they can offer functionality for iterating in time or space while executing the run()-procedures of their contained components at each iteration step.

The simplest context component in JAMS is the *Model Context (MC)* which represents a JAMS model. This component manages an ordered list of other components whose run()-procedures are sequentially executed only once. Fig. 2 shows an example of a MC containing three other components and the resulting code fragment of the MC's run()-procedure.

The *Temporal Context (TC)* is responsible for managing temporal aspects of model execution. For this

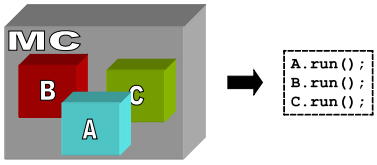

Figure 2: JAMS model context

purpose the following information have to be provided to the TC:

- a start / end date and a step size (e.g. one day) and
- an ordered list of components to be executed by that TC.

During model execution the TC will iterate over the time interval defined by start and end date and will execute the run()-procedures of all of its contained components at a time. Fig. 3 shows a TC containing again three other components and the resulting code fragment.

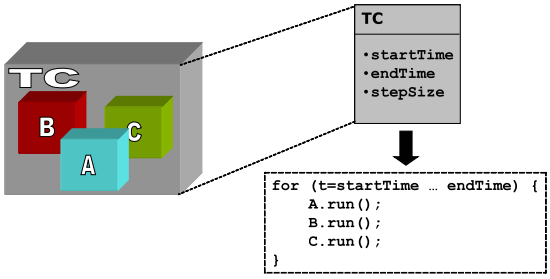

Figure 3: JAMS temporal context

The *Spatial Context (SC)* does nearly the same as the TC, but this time for the spatial domain. The information to be provided to the SC therefore are:

- an ordered list of spatial model entities and
- an ordered list of components to be executed by that SC.

Similar to the TC the SC will iterate over the specified spatial model entities and executes the run()-procedures of all contained components at a time. These spatial model entities (e.g. hydrological response units – HRUs) are represented by special objects in JAMS. These *Spatial Entities* serve as flexible repositories for arbitrary spatial attributes. The number of attributes and their values can easily be modified by JAMS components during model runtime. Fig. 4 finally shows a SC with three other components and the resulting code fragment.

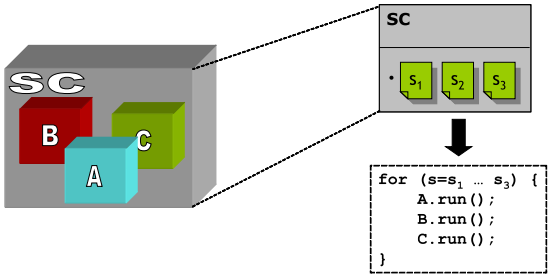

Figure 4: JAMS spatial context with spatial entities $(s_1, \ldots, s_3)$

With these context components at hand arbitrary model layouts and component hierarchies can be created. Fig. 5 shows an example of a JAMS model using all three of them. Here, the model context contains only one component: a temporal context which iterates over some time interval in monthly time steps. This monthly context in turn includes two components:

1. a spatial context iterating over some entities (nitrate response units – NRUs) and executing three components (i.e. their run()-procedures) implementing some process algorithms that are applied to the NRUs, and

2. another temporal context iterating in daily time steps within the monthly time context.

This second daily context includes two spatial contexts – one of them executing components on HRU entities and the other one executing some other components on river reach entities.

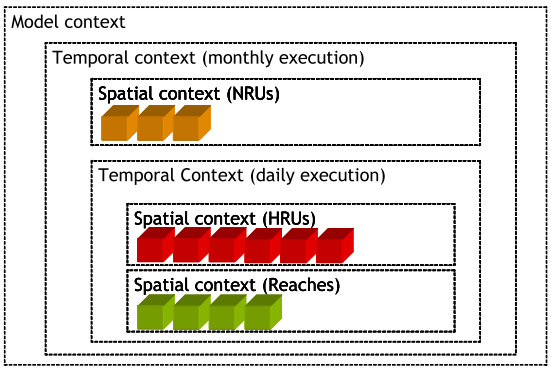

Figure 5: JAMS model example

In JAMS, all of the mentioned context components are derived from simple components. Fig. 6 shows a class hierachie overview for these different components. Here, simple components are implemented by the JAMSComponent class, and context components by the respective subclasses of the JAMSContext class. Any context component can therefore contain either simple components or other context components. This makes sure that arbitrary model layouts can be created using JAMS.

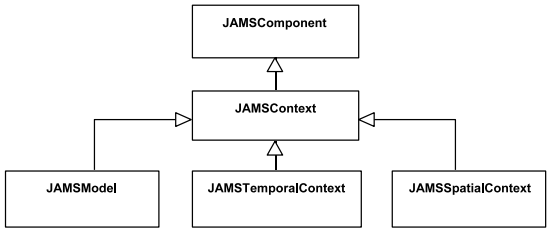

Figure 6: JAMS component hierarchy

4.3 Data Exchange

In JAMS, a component can exchange data with another component or a spatial entity provided by a spatial context. For this purpose a component has to meet two conditions:

1. At implementation time the input und output data of the component must be declared.

2. At configuration time of the whole model the sources of input data must be specified.

The declaration of a components input and output data is done with the help of JAVA annotations. These metadata are used to provide the following information about a component's attributes that are to be accessed from outside:

1. The *access type* defines if the component wants to read from this attribute, write to this attribute or both of them.

2. The *update type* defines when the component wants to access the attribute (i.e. at initialisation or at run stage). For attributes that are to be set only once at the beginning (e.g. calibration parameters) this will be at initialisation stage. The run stage access type is intended for all other attributes (i.e. those that change during model execution).

Additional metadata can be used to provide information about the purpose of a component or its single attributes, the units of attributes and an attribute's minimum and maximum allowed values. All these metadata can easily be utilised to automatically generate documentation about JAMS components.

After having declared all of a component's input and output data these can be linked to related attributes of other components or spatial entities. This linkage is being controlled by the runtime system which makes sure that all attributes are set to the correct values.

If a component is nested inside some spatial context it can access the attributes of the current spatial entity during the run stage. The runtime system keeps track of updating the values of the component's and entity's attributes depending on the access type. New attributes are automatically added to spatial entities by adding components to the model that have write access to these attributes.

4.4 Model description

In order to be executed by the runtime system a JAMS model must be described by a XML document. This document includes all necessary information for

- loading the desired components,
- creating the designated component hierarchy by nesting components within context components,
- linking components attributes to attributes of other components or spatial entities, and
- defining initial values for component attributes.

This XML-based description must be created by hand at the moment. A graphical model configuration tool that will generate this XML-document will be provided at a later point in time. Fig. 7 shows the simplified XML representation of the JAMS model sketched in Fig. 5.

```
<?xml version="1.0" encoding="utf-8"?>
<model name="Example" author="Kralisch" date="03.03.2006">

  <!-- Monthly time step -->
  <contextcomponent class="TemporalContext" name="MonthlyContext">

    <!-- NRUs -->
    <contextcomponent class="SpatialContext" name="NRUContext">
      <component class="A" name="A">...</component>
      <component class="B" name="B">...</component>
      <component class="C" name="C">...</component>
      ...
    </contextcomponent>

    <!-- Daily time step -->
    <contextcomponent class="TemporalContext" name="DailyContext">

      <!-- HRUs -->
      <contextcomponent class="SpatialContext" name="HRUContext">
        <component class="D" name="D">...</component>
        <component class="E" name="E">...</component>
        <component class="F" name="F">...</component>
        ...
      </contextcomponent>

      <!-- Reaches -->
      <contextcomponent class="SpatialContext" name="ReachContext">
        <component class="G" name="G">...</component>
        <component class="H" name="H">...</component>
        <component class="I" name="I">...</component>
        ...
      </contextcomponent>

    </contextcomponent>

  </contextcomponent>
</model>
```

Figure 7: Simplified JAMS model description

5 REFET - APPLICATION

As an example application the FAO reference evapotranspiration (refET) described by Allen et al. [1998] was implemented as a JAMS model. FAO refET calculates potential evapotranspiration for a defined surface of short grass, by a simplified form of the Penman-Monteith equation. Required input are climate variables of temperature, wind-speed, relative humidity and sunshine duration. The refET calculation itself is based on other hydro-climatic variables (net radiation, vapour pressure deficit, slope of saturation pressure curve and psychrometer constant) which have to be calculated in advance. In terms of JAMS the entire calculation procedure can be understand as a set of simple components which implement the single calculation procedures or which provide the necessary input data inside a temporal context component which iterates over time. Spatial distribution can be implemented by nesting a spatial context component between the temporal context and the process components.

For the JAMS implementation the single calculation procedures for the hydro-climatic variables are implemented as process components which provide the specific variable and get input by data reader components. The final refET process component is then provided with the necessary variables by the variable process components and the data readers. If the refET calculation is carried out for only one point the data transfer is solved by direct component interaction whereas for a spatial distributed calculation spatial entities are the receivers and providers for the calculation values. A graphical representation of the distributed JAMS-refET model is shown in fig. 8.

As an example the process which calculates the saturation vapour pressure deficit (calcVPD) needs air temperature and relative humidity as input and provides VPD as output. The XML description for this data interaction is shown in fig. 9 for the point model and the distributed realisation. Fig. 10 shows the belonging run()-procedure.

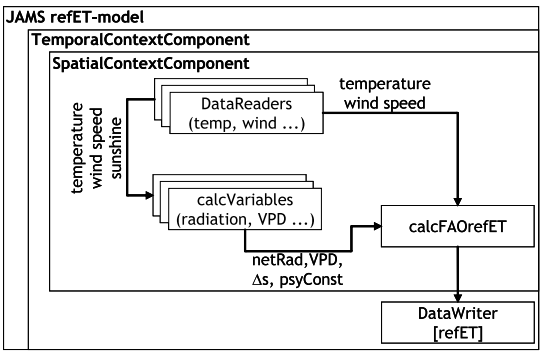

Figure 8: Graphical representation of the distributed JAMS-refET model

```
<component class="org.unijena.refET.calcVPD" name="calcVPD">
   <jamsvar name="tmean" provider="TmeanDataReader" value="tmean"/>
   <jamsvar name="rhum" provider="RhumDataReader" value="rhum"/>
   <jamsvar name="vpd" value="vpd"/>
</component>

<component class="org.unijena.refET.calcVPD" name="calcVPD">
   <jamsvar name="tmean" provider="spatialContext" value="tmean"/>
   <jamsvar name="rhum" provider="spatialContext" value="rhum"/>
   <jamsvar name="vpd" provider="spatialContext" value="vpd"/>
</component>
```

Figure 9: XML excerpt of calcVPD data interaction for point model (upper part) and distributed model (lower part)

```
public void run() {
  double esT = 0.6108 * Math.exp((17.27 * tmean.getValue())/
               (237.3 + tmean.getValue()));
  double ea = esT * (rhum.getValue() / 100.0);
  vpd.setValue(esT - ea);
}
```

Figure 10: run()-procedure of the calcVPD component

6 Conclusion and Outlook

Besides the relative simple refET model components the more complex distributed hydrological model J2K has been implemented into JAMS [Krause and Kralisch, 2005]. The experiences made during the J2K implementation showed that JAMS is suited very well to divide a model's system infrastructure from the scientific semantic of single processes. During the transfer of J2K to JAMS context components proved to be a flexible approach to create arbitrary model layouts that consider different temporal and spatial resolutions within one simulation model. Due to its modular design JAMS models can easily be adapted to new problems by exchanging single process components with newly implemented ones. Tests and comparisons with the J2K legacy implementation showed that the JAMS version did not only provide identical results but also a very competitive runtime performance.

The future work on JAMS will focus amongst others on providing graphical user interfaces for easy model configuration as well as generic tools for sensitivity and uncertainty analysis and automatic model optimisation. A transfer of solute transport models from SWAT [Arnold et al., 1998] is currently under work. Another challenge will be the extension of JAMS context components that allows a parallel execution of their contained components. Finally, extensions to easily interface relational and geo-relational databases are planned in order to provide a flexible way to deal with distributed geodata and time series.

References

Allen, R. G., L. Pereira, D. Raes, and M. Smith. *Crop evapotranspiration: Guidelines for computing crop water requirements*, volume 56 of *FAO Irrigation and Drainage Paper*. UN-FAO, Rome, Italy, 1998.

Arnold, J., R. Srinivasan, R. Muttiah, and J. Williams. Large Area Hydrologic Modelling and Assessment. Part I, Model Development. *Journal of the American Water Resource Association*, 34(1):7389, 1998.

David, O., S. L. Markstrom, K. W. Rojas, L. R. Ahuja, and I. W. Schneider. The Object Modeling System. In Ahuja, L., Ma, L., and Howell, T. A., editors, *Agricultural System Models in Field Research and Technology Transfer*, pages 317–331. Lewis Publishers, 2002.

European Union. Directive 2000/60/EC of the European Parliament and of the Council of 23 October 2000 establishing a framework for community action in the field of water policy. *Official Journal L 327*, pages 0001–0073, 2000.

Kralisch, S., P. Krause, and O. David. Using the Object Modeling System for hydrological model development and application. *Advances in Geosciences*, (4):75–81, 2005.

Krause, P. and S. Kralisch. The hydrological modelling system J2000 - knowledge core for JAMS. In Zerger, A. and lastname Argent, R. M., editors, *MODSIM 2005 International Congress on Modelling and Simulation*, pages 676–682. Modelling and Simulation Society of Australia and New Zealand, December 2005.

Leavesley, G. H., S. L. Restrepo, M. D. Markstrom, and L. G. Stannard. The Modular Modeling System (MMS). USGS, Open File Report 96-151, USGS, Denver, Colorado, 1996.