TemPSS: A service providing software parameter templates and profiles for scientific HPC

Jeremy Cohen, Peter Austing, John Darlington
Department of Computing
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
Email: jeremy.cohen@imperial.ac.uk

Chris Cantwell, David Moxey, Spencer Sherwin
Department of Aeronautics
Imperial College London
South Kensington Campus
London SW7 2AZ, UK

Jeremy Nowell, Xu Guo
Edinburgh Parallel Computing Centre
The University of Edinburgh
JCMB, Peter Guthrie Tait Road
Edinburgh EH9 3FD, UK

***Abstract*—Generating and managing input data for large-scale scientific computations has, for many classes of application, always been a challenging process. The emergence of new hardware platforms and increasingly complex scientific models compounds this problem as configuration data can change depending on the underlying hardware and properties of the computation. In this paper we present TemPSS (Templates and Profiles for Scientific Software), a web-based service for building and managing application input files in a semantically focused manner using the concepts of software parameter templates and job profiles. Many complex, distributed applications require the expertise of more than one individual to allow an application to run efficiently on different types of hardware. TemPSS supports collaborative development of application inputs through the ability to save, edit and extend job profiles that define the inputs to an application. We describe the concepts of templates and profiles and the structures that developers provide to add an application template to the TemPSS service. In addition, we detail the implementation of the service and its functionality.**

I. Introduction

High Performance Computing (HPC) platforms are now widely used across a wide range of scientific domains to undertake complex and often highly computationally-intensive processes. In recent years, computing hardware has evolved to encompass a wide range of many-core architectures and is becoming increasingly heterogeneous. This shift is also evident in modern HPC infrastructure which now provides developers with a broader range of choices for running their software. While traditional cluster platforms are widely used, other options such as computational grids and, more recently, Infrastructure-as-a-Service (IaaS) cloud computing facilities have become more widely available. These advances in hardware platforms offer developers the potential to access hitherto unseen levels of computational power, providing the capability to extend their models to tackle increasingly large and complex problems across a wide range of domains.

As both scientific models and the underlying hardware and infrastructure become more complex, so too does the range of parameters that underpin scientific codes. Inevitably, the wide range of hardware that is now available means that end-users of the software are faced with various performance-related options, many of which will be difficult to understand without a more detailed understanding of the numerical methods and how they are implemented within the code. Similarly, complex numerical models usually contain a vast array of options. Many of these can be seen as advanced; for example, they may add an additional element of stability or robustness to the software whilst not significantly altering the results that are obtained. Advanced options can therefore be extremely useful. However it is not always clear to the end-user what appropriate values for these parameters should be. This requires a detailed understanding of the software that many end-users do not necessarily possess, and they must therefore rely on more experienced users and developers.

Ultimately, the way these parameters are interpreted and handled will be a result of the understanding that the developer or developers had of the scientific task being solved at the time the code was written. Explicit parameters are straightforward for users to specify, but in complex codes with a wide array of inputs, the consequences of specifying particular groups of parameters alongside different types of input data can be hard to identify. Much of a developer's understanding of the problem being solved is lost in the process of converting their solution into code. We believe that by representing this information in a form of structured metadata, users can be helped in understanding the software and efficiently specifying their job inputs and developers can be assisted in maintaining and extending the code.

To address these challenges, in this paper we detail our model of software parameter templates and job profiles that were initially introduced in [1] and introduce TemPSS (Templates and Profiles for Scientific Software), a service that supports the development and use of templates and profiles. TemPSS has been developed as part of the libhpc framework [2] which is being built to provide an environment to support the simplified specification and execution of scientific applications on heterogeneous computational resources [3]. We use templates and profiles to provide a means of representing software parameters in a structured manner and describe how they are defined by developers and used by an application's end users. The TemPSS service [4], which is available on GitHub, consists of a web service and client-side JavaScript library to handle the management of templates and their use to create profiles which the service then transforms into application input files or parameters. The service may be used standalone but is designed to be able to be integrated

with other frameworks that support the configuration of HPC software. To support standalone usage, TemPSS comes with a prototype web-based user interface for creating, editing and storing profiles.

In Section II, we describe the concepts of software parameter templates and job profiles in detail and explain their structure and aims. We then look at related work in Section III and describe the developer-focused processes of defining templates and producing the necessary transforms to generate application input data from a job profile in Section IV. Section V describes the TemPSS service itself, its design, implementation, API and usage. We present conclusions and future work in Section VI.

II. Software Parameter Templates and Job Profiles

Software parameter templates and job profiles are metadata structures that are designed to represent application configuration parameters and users' job specifications in a manner that makes them straightforward to understand and work with. They were introduced in [1] and we define them as follows:

- **Software parameter templates** provide an explicit representation of the decision-space and parameters that exist within an application. These properties are stored with associated metadata and can be grouped according to their semantic relationship within the application. A template does not define any specific values for the properties it contains, it only includes a definition of parameters, including constraints on their values, their datatypes, documentation and other related metadata.
- **Job profiles** are structures that provide instantiations of templates. A profile assigns values to one or more of the properties defined in a template. If a profile contains values for a minimum set of required template parameters that are necessary to run the underlying application, the profile is said to be a *valid profile*. If a profile contains only a subset of required values, it is said to be a *partial profile*.

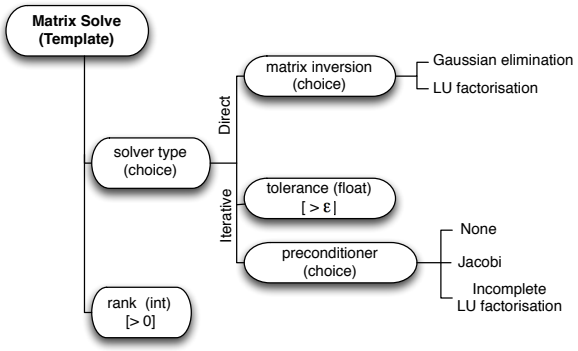

Fig. 1. Example of a simple template structure containing nodes with different data types.

A template consists of a root node, labelled with the template's name. Any node can have one or more child nodes and each node has a type, such as a choice or a basic datatype

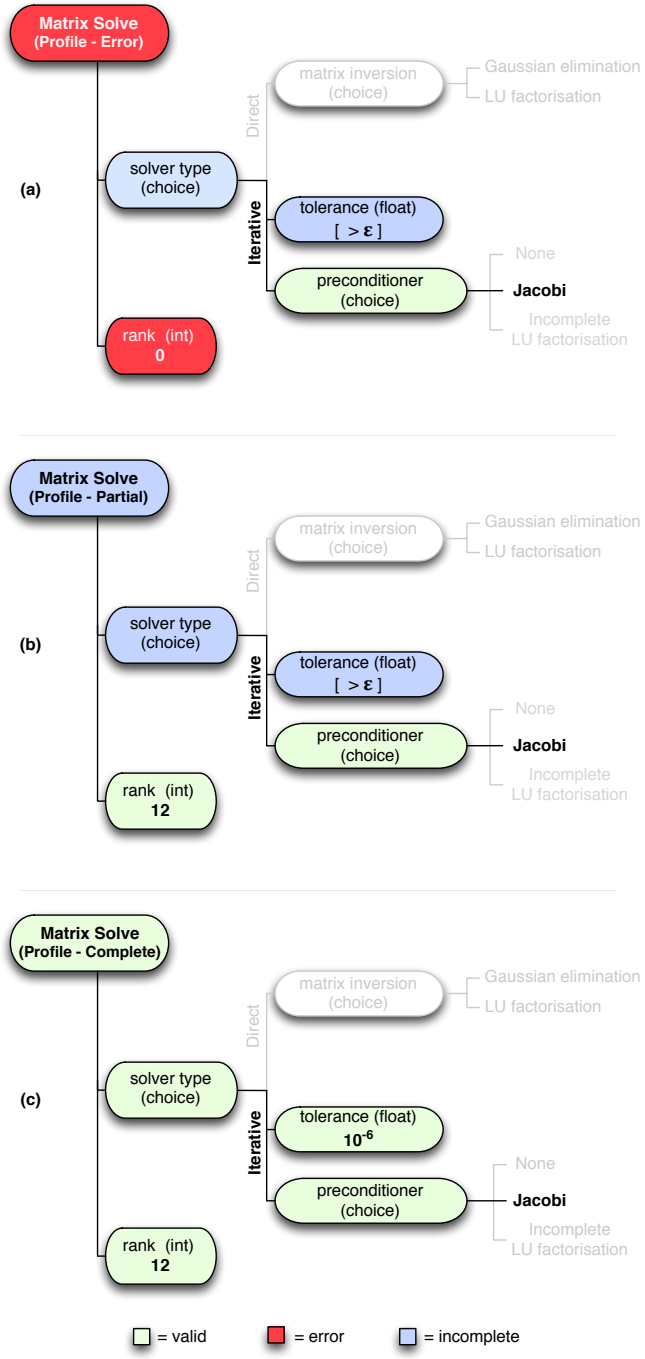

Fig. 2. Three possible template instantiations are shown in Figure 1. (a) shows a partially complete profile in an error state, (b) shows a partially complete profile and (c) shows a complete and valid profile.

(e.g. `string`, `int`, `double`), or a category type which acts to group a series of other nodes. A choice node displays a drop-down box showing the possible values that may be selected. The selection of a value determines the child nodes that will be shown for this node. Basic type nodes display a text input box in which the user can enter a value of the correct type. Validation is carried out on values entered for basic type nodes and an error is displayed if a value of the wrong type is entered.

Rather than providing a direct representation of application input files or command line parameters, some properties in a template may not map directly to any one input parameter but instead represent complex groupings of values within application input data. This semantically focused representation of properties helps to provide a more straightforward approach to software configuration.

Figure 1 shows an example of a simple template representing a subset of properties from an application to solve a linear system. The template consists of two values at the first level of the tree - `solver type` and `rank`. `solver type` is a choice element from which the user can select one of two options – `Direct` or `Iterative`. Selecting `Direct` will result in the user having to provide a value for `matrix inversion`, another choice element, while selecting `Iterative` as the solver type will result in the user needing to specify values for `tolerance` and `preconditioner`. `rank` is required to be an integer and has the constraint that the provided value must be greater than 0. `tolerance` must be a float value and has the constraint that the provided value must be $> \epsilon$. Realistic templates are, of course, likely to be significantly more complex than this but this small-scale example is designed to demonstrate how a template is structured and how it can be instantiated as a partially or fully complete profile.

Figure 2 shows three possible instantiations of the template structure from Figure 1. Figure 2(a) shows a partially complete profile that has an error due to the value for `rank` not meeting the constraint that the value must be greater than 0. The `solver type` property is shown as incomplete because not all of its sub nodes have values assigned. Figure 2(b) shows a partially complete profile with the error condition from (a) resolved by setting `rank` to 12. The top level node still shows the profile as being incomplete, however, since one of its child nodes (`solver type`) is incomplete. This is, itself, a result of one of the child nodes of `solver type` not having a value assigned. This demonstrates how the state of any incomplete node in a tree propagates up to the top level resulting in the tree representing a partial profile. The same propagation can be seen in the case of an error state in (a) where the error with the `rank` property results in the entire profile being in an error state. After assigning a valid value (10^{-6}) to `tolerance`, Figure 2(c) shows a complete and valid profile which could then be used to generate input data for the target application through processing with the template's associated Extensible Stylesheet Language Transformation (XSLT) [5] document. An XSLT processor reads the XML document representing the profile and applies the transformation defined in the XSLT document to generate an application input file or data representing a set of command line parameters for the target application.

The template and profile model has been developed alongside the libhpc framework [2], [6] which has been designed to offer an environment for the specification, deployment and execution of HPC jobs on heterogeneous resources. Templates and profiles represent one element of the job specification layer in libhpc which takes a user-focused approach to simplifying HPC job execution. In Section V, when presenting the TemPSS service, we describe how the template and profile functionality has been integrated into one of libhpc's exemplar applications, the Nekkloud [7] web interface, for running high-order finite element simulations via the Nektar++ spectral/hp element framework [8].

III. Related work

There are various examples of user interfaces designed to simplify access to, and configuration of, complex scientific software across a wide range of domains. An early example of a toolkit for developing web-based or standalone graphical interfaces for scientific software is Javamatic [9]. Built in Java, this organises a collection of software components (executables) relating to an application (workflow) into a hierarchical tree of categories allowing the user to input parameters used by multiple components at a common node. The web-based interface allows execution of the software remotely via the web-browser. A similar functionality is provided by a web-based interface for molecular biology [10] which provides a homogeneous interface to multiple programs, suppressing syntactic and semantic complexity by providing context-based access to documentation.

More recently, Rappture [11] has been developed, which automatically generates standalone user interfaces for scientific software, in a variety of programming languages based on an XML definition of possible input parameters. Default values can be provided, but no constraints between parameters can be imposed. The GUI [12] developed for Zori, a quantum Monte Carlo program, is built on Rappture and extends its capabilities to provide parameters for a linear sequence of separate software components, and to dynamically present parameter choices depending on the algorithm selection. It also includes deployment to PBS clusters as well as local execution. Another example of a domain-specific web interface is ScattPort [13], which provides a single interface for users to configure and run light scattering experiments using a vast range of available simulation tools.

The interfaces described so far provide minimal validation of the values entered by the user, particularly in terms of inter-parameter constraints. This aspect is essential for complex simulation codes, such as those used in ocean modelling, where a web-based interface, LEGEND, has been designed [14] for job specification and execution. An XML mark-up language, LCML [15], is used to describe the compile-time and run-time parameters needed to generate and execute a simulation. Parameters are grouped into categories (e.g. by program, compilation/runtime, makefile parameters, stdin parameters) and are validated (with checks for type, range and conflicts with other parameters). In contrast to our approach here, support for compile-time options is included, as an alternative to the software component model used in libhpc. FoSSI [16] focuses on providing a simple end-user programming interface for a range of solvers and relying on a domain expert to maintain and extend the interface to support new software. This is similar to our approach of requiring experienced domain scientists or developers to develop the template structures in TemPSS while providing a straightforward interface to the end-user. TAPM [17], an air pollution modelling application, provides a user interface for model configuration. To support rapid setup of a model by a user, it also provides datasets of key simulation inputs, a similar method to our use of profiles to store job configurations. Recent demand for simplified access

to scientific applications is illustrated by the recent emergence of a company providing user-friendly interfaces to scientific software [18].

Focusing more on metadata and the information that defines and supports computational models, Velo [19] is a knowledge-management framework based on a wiki environment. It provides management of input data and the results of simulations and supports the running of simulations on remote resources. Work described in [20] on the PRIMA platform demonstrates how bringing together Velo with a workflow environment, MeDICI, combines the capabilities of the two systems allowing running and real-time monitoring of workflows through via Velo environment.

Existing web-based user interface solutions already provide a domain-focused interface to complex scientific software. However, parameters are often presented as a flat list or collated into broad categories and validation of parameter values against constraints is mostly minimal. This aspect has been considered in greater detail in relation to software component testing, where parameters, their possible values and their interdependencies are mapped out through a directed graph [21]. This ensures all valid (and only valid) parameter combinations can be tested. The graph represents the input parameters as nodes and possible values on the edges. However, parameters are evaluated in a fixed order. To our knowledge none of these interfaces provide the collaborative approach to job specification we describe here.

IV. Defining templates and transforms

In order for TemPSS to provide users with a template for a particular application, the underlying template metadata must be specified. Further, to allow profiles, which are instantiations of templates, to be useful to an application user, mappings or transforms must be defined that convert a complete and valid profile to one or more input files required by an application. This work should be undertaken by someone familiar with the implementation details of the application in question, perhaps an application developer. Ideally the information required to produce the template metadata and transformation should be recorded as the application is developed. However, a wide array of existing scientific applications are potential targets for TemPSS and for these applications, the required metadata is not necessarily readily available.

Instead, analysis of an application's input data, and where required, the application code itself, must be undertaken to identify decisions that take place during application execution and their associated configuration options and descriptions. It is then necessary to identify how best to represent these properties in metadata. In addition to representing standard input parameters, the aim of this process is to identify how different parameter groupings or values can affect application execution with respect to different input data or target hardware platforms. Templates and their associated transforms can then be structured to offer users more intuitive configuration options that may not always represent a direct mapping to one value in the resulting application input data. The template specification process has been undertaken for a group of solvers from Nektar++ [8], a high-order spectral/hp element framework, and for a subset of tools within the GROMACS molecular dynamics software [22], [23]. The process of specifying templates is now described, along with details of how the metadata and transforms are defined.

A. Template specification

Nektar++ provides a collection of libraries that implement the spectral/hp element method, and a set of solvers that implement a number of partial differential equations including the incompressible and compressible Navier-Stokes equations and the monodomain model for simulating cardiac electrophysiology. Templates for a set of the most widely used solvers in Nektar++ have been developed. As described in Section II, TemPSS templates are displayed as trees and a major part of the template specification process is deciding the best way to group and present application parameters within the tree structure. The way that parameters are grouped is important for two reasons. Firstly, to support collaborative profile development, parameters should be grouped according to their relevance to different stakeholders, for example, all parameters related to the target hardware platform should ideally be shown together. Secondly, from the end-user perspective, a poorly structured set of parameters may be confusing and end up making the job specification process less intuitive. Developing application templates requires an understanding of the many input parameters and the relationship between them, and of the internal processes of an application such as the algorithms that are undertaken and the way they are implemented. In the case of Nektar++ the template development process was performed by a TemPSS developer, and required a detailed study of the Nektar++ configuration files, the underlying application code itself, and many helpful interactions with the Nektar++ developers. The result is a set of templates which are clearly defined, grouping related parameters, and providing a means to understand both the interface to the application and therefore the application itself.

For GROMACS, a slightly different approach was taken. GROMACS was chosen because of existing links with users of the software, its wide use within the molecular dynamics community and because it consists of many different components, each with their own set of inputs, providing a means of experimentation with linking different component templates together. To drive the template development process a small-scale, but realistic, protein simulation example was chosen. A full template was developed for the main simulation component (mdrun) and minimal templates were developed for the other components in the example in order to enable a complete run. The GROMACS template development required study of the GROMACS online manual [24], which provides detailed information on input parameters. The intention is to develop the structure of the templates further based on user feedback.

The current manual process of template development, if undertaken thoroughly and comprehensively, represents significant overhead for developers. However, where scientific applications are complex but widely used, it is believed that the investment of time in undertaking this process is reasonable given the potential benefit to existing users and the opportunity to target the application at a wider range of new users. Structuring an application's metadata, such as the information contained within the GROMACS manual referred to above,

into a form that encompasses the expert knowledge of the developers who built the application, offers the opportunity to present the information in a way that should make it understandable by anyone with even a cursory knowledge of the domain. We could even conjecture that the way developers think through the process of mapping knowledge to code is in the form of a tree, based on a series of statements of the form "*if A then B*". As a result, mapping this knowledge into a visual representation could help both users, who can obtain information in a clear form without having to request the assistance of developers, and the developers themselves, by providing them with a representation of their code in a manner that makes it easier for them to support and maintain it. New users may understand the scientific processes undertaken by an application but not have the technical computing knowledge or experience to build their own input files from scratch. In such cases, the ability to interact with a visual interface providing context-specific documentation and input validation can help end-users to undertake jobs that may have previously required support from computer scientists and hardware providers.

B. Template implementation

Implementation of a template structure requires the specification of each input parameter within the tree, and the ability to annotate parameters with, for example, documentation or units of measurement. The choice for this task was XML Schema Definition (XSD) language [25], [26]. XML Schema provides a series of structures and datatypes that are used to provide the definition of template parameters. A *simple type* is used to describe an individual parameter, while a *complex type* is used to link parameters together to form branches of the tree. Use may be made of XML Schema's primitive data types to define a parameter, such as a *float* or *string*, and these may be extended to construct new types by, for example, placing a restriction on the range of allowed values, for instance specifying that a *float* must be positive, or providing an enumeration of choices. As an example, a strictly-positive double precision number can be defined as follows:

```
<xs:simpleType name="positiveDouble">
  <xs:restriction base="xs:double">
    <xs:minExclusive value="0"/>
  </xs:restriction>
<xs:simpleType>
```

Validation of data entered into a template tree can then be carried out based on the types used in the template schema to ensure that provided data fits within a required range or set of values.

XML Schema allows for the inclusion of one schema document within another provided the documents are within the same namespace, therefore it is possible to build, for example, a common GROMACS schema which defines a core set of parameters used in several different components. This common schema can then be included by specific component template schemas which can compose both the common parameters and any locally-defined component-specific parameters into template tree definitions, as required. An example of a type from a Nektar++ solver template which makes use of the `positiveDouble` type shown above and is annotated with documentation and units is:

```
<xs:complexType name="Monodomain">
  <xs:sequence>
    <xs:element name="Chi" type="positiveDouble">
      <xs:annotation>
        <xs:appinfo>
          <libhpc:documentation>
            Membrane surface to volume ratio
          </libhpc:documentation>
          <libhpc:units>
            mm&lt;sup&gt;-1&lt;/sup&gt;
          </libhpc:units>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <...>
```

Specifying templates in XML Schema enables instantiations of these templates as profiles to be represented using XML documents which may then be validated against the template schema on which they are based in order to ensure that they have a correct structure.

C. Transforms

Templates provide a means of specifying an application's configuration and input parameters, and allow profiles to be built for different application use cases. However, for these profiles to be useful they have to be translated into the input format of the application itself. Thus transformations are required to convert between a profile and the means of job input and configuration used by the target application. The obvious choice for this is XSLT [5]. An XSLT transformation enables XML profiles to easily be transformed into any format required by the target application. For instance in the case of Nektar++ where the software uses an XML input file format, the transformation input is the XML profile and the output is a Nektar++ XML input file. The use of transforms means that a profile can define a different, user-focused, structure to the input file for a target application. This structure may even include different parameters that are converted into application-supported parameters as part of the transform. The resulting application input file(s) may be in a very different format to the structure used in the profile. An example of a small section of an XSLT transform relating to the `Chi` parameter used in the Nektar++ cardiac electrophysiology solver is shown below:

```
<xsl:template match="Physics" mode="CardiacParameters">
  <xsl:if test="Model/Monodomain">
    <P>
      Chi = <xsl:value-of select="Model/Monodomain/Chi"/>
    </P>
  </xsl:if>
</xsl:template>
```

This example shows how the transform is structured to allow the XSLT processor to match profile elements as it moves through the profile document. When the `Monodomain` parameter that is a subset of `Model` within the `Physics` block is found, the processor outputs the specified string, in this case a `<P>` XML tag containing `Chi =` followed by the value of `Chi` that is stored in the profile. It then closes the tag by outputting `</P>`.

In contrast, for GROMACS, profiles are converted into a simple text format defining a series of command line parameters that can be used as the command line input to GROMACS components.

A further use of XSLT within TemPSS is the conversion of template schema documents to HTML. This is done by an embedded XSLT transform that reads template schema documents and converts them into the HTML tree structure that can be seen in Figures 5-8. The TemPSS API can be used by third parties to obtain this HTML tree structure from a template such that it can be used within third-party user interfaces that make use of the TemPSS service.

V. The TemPSS service

The TemPSS service is a web service that integrates the functionality described in the earlier sections of this paper into an open source tool that can be built and run locally or deployed on a remote server for shared access. Code for the service [4] is available on GitHub. TemPSS provides a combination of server-side functionality, made available to developers via a REST API, and a client-side JavaScript library for working with templates and profiles and designed to assist with integrating TemPSS into another service or application. End-users can access the system through a web-based profile manager interface provided with the service. Alternatively, third-party developers may choose to build their own user interfaces that build on top of the service's REST interface and client-side JavaScript libraries. TemPSS's functionality covers three key areas:

- Template specification and use
- Profile generation and management
- Application input file generation

In this section we describe the structure, functionality and use of the TemPSS service. We look, in turn, at service architecture and implementation, configuration, the REST API, the client-side JavaScript library and the web-based profile manager and editor.

A. Service Implementation

TemPSS is a Java web application and makes use of a number of third-party libraries. The code is compiled and packaged using the Apache Maven project build tool that handles obtaining all the necessary dependencies and packaging the application for deployment onto a web application server. The service has been developed and tested with the Apache Tomcat application server. Figure 3 shows the system architecture. The application configuration specifies three Java servlets. One provides the original HTTP POST interface to the service that has been retained for use by any clients that prefer this means of accessing the service over the REST API. The other two servlets are the SpringMVC dispatcher servlet that handles requests to the application's web interface and the Apache Jersey servlet that handles REST API requests.

Apache Jersey is the reference implementation of the Java API for RESTful Services (JAX-RS) specification. It receives incoming HTTP requests and directs them to the relevant Java function within the service implementation based on various request properties including the request type, the content type of any provided data and the accepted response type(s) specified by the client. XSLT processing is handled by Java's API for XML Processing (JAXP) which is bundled with the Java Standard Edition library.

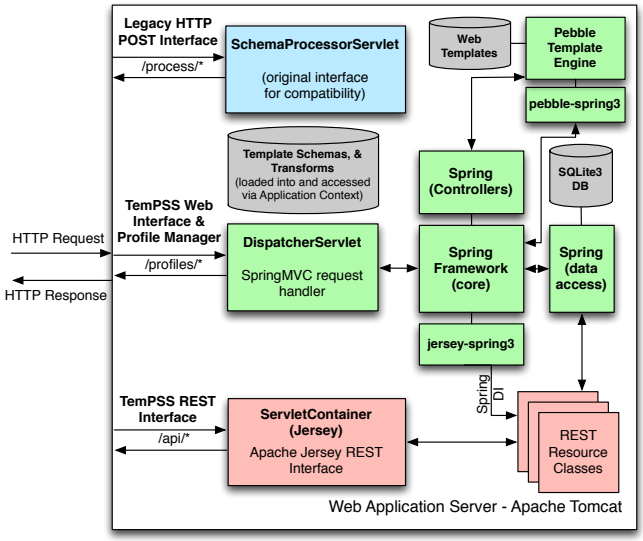

Fig. 3. The architecture of the TemPSS application.

The Spring Framework is used to provide dependency injection and the MVC element of Spring is used to support the web interface. Spring's data access support is used, via JDBC, to interface with a SQLite3 database that stores profiles. We inject a reference to the data access layer into the REST classes so Apache Jersey needs to be aware of, and integrate with, Spring for which we use the jersey-spring3 library. Web pages are built as templates using the Pebble template engine [27] which also provides a support library for integration with Spring. Specifically, Pebble provides a Spring *view resolver* that allows Spring MVC to hand view requests over to Pebble for resolution. The interactive tree display, which can be seen in Figures 5-8 and is used for our template editor, is based on a bootstrap tree widget [28]. On to this tree design, we have overlaid the validation symbols that are used to show if a value entered into the tree is valid or invalid.

B. Service configuration

At service startup, TemPSS looks for template configuration information which defines the templates that will be available to service users. Section IV has detailed the low-level specification of template structures using XML Schema and the development of XSLT transforms to convert profiles for a particular template into valid and complete application input files for the target application. Once a developer has prepared this material, they need to register their template(s) with the TemPSS service. This is done by writing a property file which must have a `.properties` extension and be structured as follows:

```
component.id=<comp_id>
<comp_id>.name="Component name"
<comp_id>.schema=<schema file name>
<comp_id>.transform=<XSLT transform file>
```

At service startup, TemPSS looks for all the property files in its template directory. The `<comp_id>` value in the sample above must be replaced with an identifier that is an

alphanumeric string with no spaces. This identifier is then used to read the component's name property – a longer or more detailed name than the identifier – and the schema and transform filename properties. The schema and transform files are, in turn, read in from the service's schema and transform directories. Setting up the property files and placing them and the schema and transform files into the relevant directories in the software distribution is the only required configuration task. The service can then be re-packaged using the Apache Maven tool and deployed to a web application server.

A property file may contain details of multiple components. This is achieved by providing a single `component.id` property, the value of which is a comma-separated list of component IDs. Separate name, schema and transform properties are then provided for each component ID.

C. REST Interface

TemPSS's REST interface allows direct programmatic access to the service's functionality and is also used by the web-based profile manager interface. The REST interface provides access to a series of template and profile-related operations. The service's full REST API documentation is available online [29] and an overview of the API functionality is now provided:

- **Template metadata:** Provides access to template metadata for the templates that have been registered with the system. The available data consists of template ID and name and the filenames for the schema and transform files. Data can be retrieved in a plaintext format or as JSON objects.
- **Template names and IDs:** Provides access to a list of either template names or template IDs as plaintext or in JSON format.
- **HTML template:** Obtain an HTML representation of a template. This is generated by passing the template's schema file through the service's built-in schema-to-HTML XSLT transform to generate an HTML tree version of the template that can be visualised in a user's browser.
- **Profile metadata:** Provides access to the names of stored profiles based on the identifier of the template that they correspond to. This information may be retrieved as plaintext or in JSON format.
- **Profile access, storage and removal:** Save, load or delete a profile that has been built in the web-based profile editor. Profiles are represented using XML data that is stored in a SQLite3 database embedded within the application.
- **Convert profile to input file:** Convert a provided XML profile document into an input file for the target application. This uses the XSLT transform registered for the application/component in the TemPSS configuration data to convert the profile into input data formatted in an application-specific manner. This transform process may also be used to generate command line input parameters.

The REST API is used by the TemPSS web-based profile manager interface to access the service's functionality but it is also designed to be used by third-party developers who want to integrate the application parameter template and job profile model into their applications.

D. Client-side Library

TemPSS's client-side JavaScript library provides functions to support common template and profile related operations that may be undertaken in client-side code. The library is used in the template manager interface but is also designed to be used by third-party developers when integrating functionality from TemPSS into their own applications. To make use of the library, developers must load two JavaScript files in their code – `libhpc-parameter-tree.js` and `tempss-manager.js`. These files, in turn, have dependencies on three additional third-party libraries and full details are provided in the TemPSS documentation. The first JavaScript file includes only client-side functionality specific to parameter trees and working with them in a web interface. This includes `collapseTree` to collapse a partially expanded tree so that only the root node is shown and `expandTree` to expand a tree to show all accessible nodes. It also provides a function `getProfileXML` to generate an XML profile from a fully or partially configured template tree. This XML can then be stored, for example to a database. An XML profile document can subsequently be loaded back into an empty template tree of the same type using the `loadLibhpcProfile` function. Some additional functions, including validation-related functions used to check the validity of values entered into a template against the template schema, are intended to be used internally within TemPSS but may also be of use to third party developers building on TemPSS. The second JavaScript file, `tempss-manager.js` provides functions related to template access and the management of profiles, both of which require interaction with an instance of the TemPSS web service. This part of the library provides the ability to get a list of template metadata from a TemPSS service instance and to get an HTML tree representation of a template from the service. It also provides the ability to save, load and delete profiles stored within a remote TemPSS service instance and to reload and reset a tree displayed on the client side to remove any values that have been entered into it.

E. TemPSS Web-based Interface

TemPSS's web-based profile manager interface is designed to provide an example of the service's functionality and to demonstrate to developers how this functionality can be accessed and used. It is also designed to be used by end-users for building, saving and editing profiles. Figure 4 shows the page that is displayed when first visiting the profile manager interface.

When the web page is opened, the interface loads the list of registered templates from the TemPSS service and a user can then select a template which is loaded into the profile editor pane. The template is shown as a clickable tree where nodes can be expanded by clicking on them. Nodes that require a selection from a small set of fixed options display a drop-down list while nodes that require the user to enter information display a text entry box. Figure 5 shows the interface with

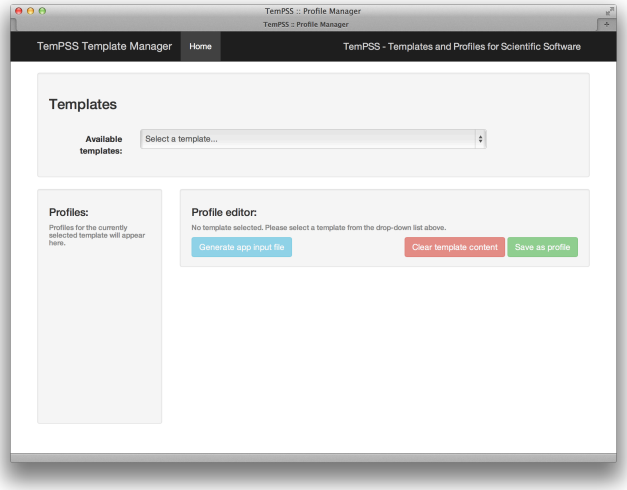

Fig. 4. The TemPSS profile manager interface screen.

a template for the Nektar++ cardiac electrophysiology solver loaded. Part of the *Physics* branch of the tree is expanded showing examples of choice nodes where selections can be made from a drop-down list and a text input node. The profile list on the left-hand pane is populated via the REST API when the user selects a template from the *Available templates* list in the top pane and shows only the profiles that relate to the currently selected template. The *Save as profile* and *Clear template content* buttons become active once a template is displayed in the profile editor but the *Generate app input file* button remains inactive until the root node of the tree in the profile editor becomes valid. The TemPSS JavaScript library generates events on nodes when they change state between valid and invalid and, in addition to being used by the TemPSS library, these events may also be of use to third-parties when integrating TemPSS and profile editing functionality into their own applications.

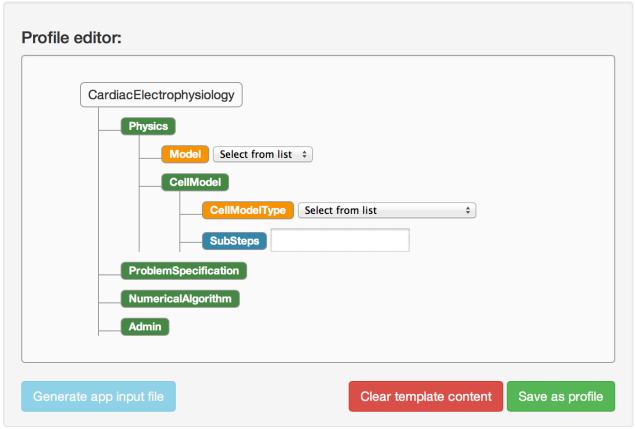

Fig. 5. The profile editor panel of the profile manager displaying a template for the Nektar++ cardiac electrophysiology solver.

As described in Section IV, documentation and validation parameters can be assigned to template nodes. When a user places their mouse pointer over a node that provides documentation, this information appears as a tooltip to guide the user on how to complete that element of the tree. After data is entered for a text input node that allows entry of numeric or string values, the entered data is checked against any validation constraints provided with the template. A visual cue, a green tick, is added to the node if the data entered is valid. Figure 6 shows a section of the web-based interface with documentation displayed and icons to identify nodes as being valid. When all the nodes in a particular branch of a template tree are valid, the parent node of that branch also becomes valid.

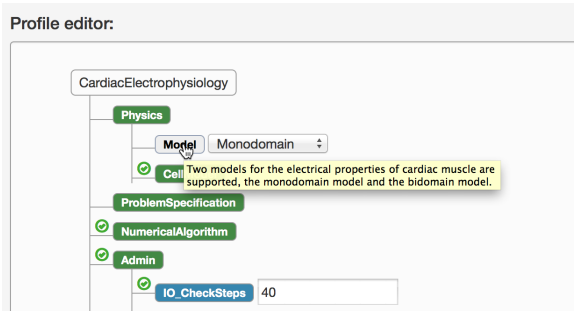

Fig. 6. Section of the TemPSS web interface showing documentation displayed for a node and a group of valid nodes.

When invalid data is entered into a field, a red '*x*' is displayed to the left of the field containing invalid data and and an icon appears to the right of the field. Placing the mouse over this icon displays a tooltip explaining the problem with the entered data. Figure 7 shows how validation errors are displayed along with information for the user to rectify the problem.

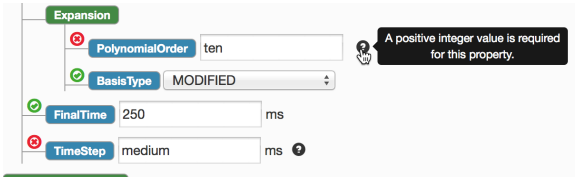

Fig. 7. Section of a template showing display of validation errors and the information shown to the user explaining the error.

Content that has been entered into a template can be saved as a profile for which the user is prompted to enter a name. When a template is selected, all stored profiles for that template are shown in the profile panel of the interface. Each profile listed in this panel provides icons to delete the profile or load it into the template, permitting subsequent extension of the profile and saving, either under a new name or under the original name. Once the top level node of a profile is shown as valid, the profile may be used to generate an application input file. This can be done by clicking the "Generate app input file" button below the template tree. The generated input file is then downloaded to the user's browser for use in the target application (see Figure 8). As demonstrated in the Nekkloud [7] web-based environment for the Nektar++ finite element software, when TemPSS is integrated into another application, the generated input file may be used to directly execute the application without returning the file to the user to run the application manually.

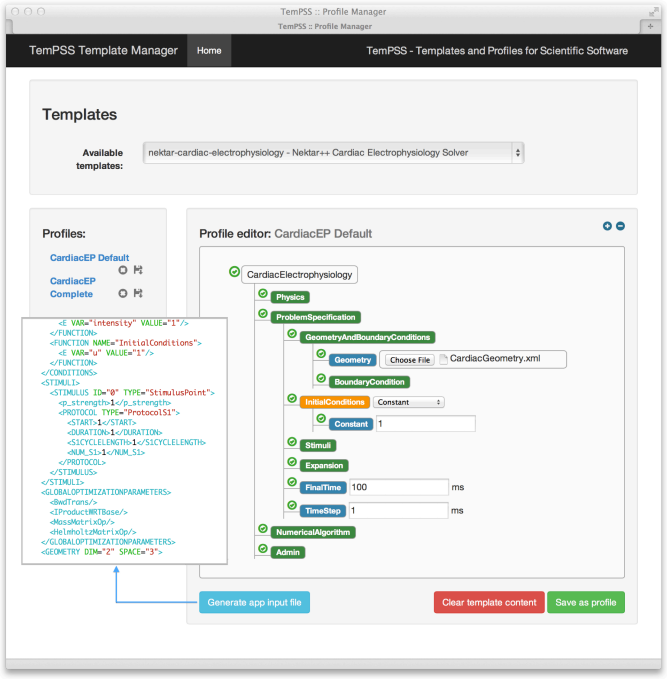

Fig. 8. The profile manager interface showing a completed profile that has been submitted and an overlay showing part of the job input file returned by the TemPSS service.

VI. Conclusion

In this paper, we have presented TemPSS, a web-based service designed to simplify the process of building and managing application configuration data for scientific applications. One of the primary goals of TemPSS is to address the challenge of increasingly complex choices in scientific software, particularly as hardware platforms evolve and become more heterogeneous. We believe that the concepts of software parameter templates and job profiles go a considerable way towards achieving this goal. Templates represent the application configuration parameters in a semantically focused tree structure, which gives end-users a clear overview of the range of possible choices, clear, context-specific documentation for these options and a way to validate their inputs before they submit their job. On the other hand, profiles are designed to encourage collaborative development of input parameters. Experienced users and developers can design partially completed profiles that, for example, choose optimal performance settings for a range of hardware architectures whilst leaving parameters specifying the simulation-specific settings incomplete.

This two-pronged approach to input configuration has several advantages for both developers and users of scientific software, as well as for the maintainers of infrastructure. From the developer's perspective, providing support for their software can be made substantially easier, both through providing incomplete profiles which clearly delineate advanced options from standard simulation parameters and by supplying a documented set of available parameter choices. Furthermore, the modular nature of TemPSS and its use of widely-supported technologies such as XML schemas means that it is relatively straightforward to extend the service to support new applications. For end-users, the interactive, web-based interface is easy-to-use and clearer than the traditional text-based input files. The purpose of each parameter is made clear from the supplied documentation, and validation takes place in real-time so that users can immediately correct any mistakes made when entering values, as opposed to waiting for the simulation to start (after potentially many hours in a queue) before realising there is an error. When building on incomplete profiles, users should be able to achieve better performance with their simulations without necessarily concerning themselves with advanced performance-related parameters which have been previously set by the developers. This therefore has an impact on maintainers of the underlying HPC infrastructure, who with increased efficiency should see more throughput of jobs and thus better overall usage of their hardware.

Our planned future work on TemPSS involves developments at the conceptual level and enhancements to the implementation. The issue that we consider may offer the greatest potential for enhanced user interaction is a modification to the tree-based structure that is presently being used to represent templates. The tree approach provides structures that are straightforward to visualise, understand and work with for users but does not, directly, support the representation of cross-tree dependencies in parameter choices, although such information could be represented in higher-level metadata. An alternative means of representing the template structure would be a graph. For visualisation, the choice of a tree structure was made over a graph due to its straightforward presentation of available choices to the end user. However, we anticipate that the need to handle cross-tree dependencies may arise in a wider range of applications than those considered here. While maintaining the visual tree, the additional flexibility of an underlying graph structure would assist in representing and handling dependencies between nodes. We consider that replacement of the underlying template structure with a graph, which is subsequently presented to the user as a dynamically-updated tree, could offer users greater flexibility. From a technical aspect, XSLT provides a powerful way to perform the transform from XML Schema to application input file, but it can be somewhat difficult for developers to use. We are considering options to either automate, to some extent, the generation of transforms, or to provide a suitable alternative means to generate job input files from the tree structure. Finally, we aim to enhance our support for extremely large files, particularly input files. For example, in our Nektar++ exemplar application, very large meshes can produce input XML files in excess of 100MB that are difficult and expensive to parse. We aim to solve this issue by producing client-side utilities to extract only the necessary portions of input data, which can subsequently be passed to the TemPSS service.

Acknowledgment

The authors would like to thank the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the projects libhpc: Intelligent Component-based Development of HPC Applications (EP/I030239/1) and libhpc Stage II: A Long-term Solution for the Usability, Maintainability and Sustainability of HPC Software (EP/K038788/1) that have supported this research.

Peter Austing was with the Department of Computing, Imperial College London when his work on templates and the

schema service was undertaken.

REFERENCES

[1] J. Cohen, D. Moxey, C. D. Cantwell, P. Austing, J. Darlington, and S. J. Sherwin, "POSITION PAPER: Ensuring an effective user experience when managing and running scientific HPC software," in *2015 International Workshop on Software Engineering for High Performance Computing in Science (SE4HPCS), ICSE '15*, Florence, Italy, 2015, (to appear).

[2] libhpc: Intelligent Component-based Development of HPC Applications. http://www.imperial.ac.uk/lesc/projects/libhpc. Accessed 11^{th} Feb 2015.

[3] J. Cohen, C. Cantwell, N. C. Hong, D. Moxey, M. Illingworth, A. Turner, J. Darlington, and S. Sherwin, "Simplifying the Development, Use and Sustainability of HPC Software," *Journal of Open Research Software*, vol. 2, no. 1:e16, pp. 1–6, 2014, DOI: http://dx.doi.org/10.5334/jors.az.

[4] GitHub – TemPSS: A template and profile manager and editor for handling application inputs for HPC software. https://github.com/london-escience/tempss. Accessed 12^{th} Mar 2015.

[5] M. Kay (Ed), "XSL Transformations (XSLT) Version 2.0," W3C, Tech. Rep., January 2007, available at http://www.w3.org/TR/xslt20.

[6] J. Cohen, J. Darlington, B. Fuchs, D. Moxey, C. Cantwell, P. Burovskiy, S. Sherwin, and N. C. Hong, "libhpc: Software sustainability and reuse through metadata preservation," in *First Workshop on Maintainable Software Practices in e-Science*, Chicago, IL, USA, Oct. 2012, position paper. [Online]. Available: http://www.software.ac.uk/sites/default/files/softwarepractice2012_submission_8.pdf

[7] J. Cohen, D. Moxey, C. Cantwell, P. Burovskiy, J. Darlington, and S. J. Sherwin, "Nekkloud: A software environment for high-order finite element analysis on clusters and clouds," in *IEEE International Conference on Cluster Computing*. IEEE, 2013, Poster paper. DOI: http://dx.doi.org/10.1109/CLUSTER.2013.6702616.

[8] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. de Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin, "Nektar++: An open-source spectral/hp element framework," *Computer Physics Communications*, vol. (Accepted), 2015, DOI: http://dx.doi.org/10.1016/j.cpc.2015.02.008.

[9] C. Phanouriou and M. Abrams, "Transforming command-line driven systems to web applications," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1497–1505, 1997, DOI: http://dx.doi.org/10.1016/S0169-7552(97)00049-4.

[10] C. Letondal, "A Web interface generator for molecular biology programs in Unix," *Bioinformatics*, vol. 17, no. 1, pp. 73–82, 2001, DOI: http://dx.doi.org/10.1093/bioinformatics/17.1.73.

[11] Rappture Toolkit. http://rappture.org.

[12] R. Olivares-Amaya, R. Salomon-Ferrer, W. A. Lester Jr, and C. Amador-Bedolla, "Creating a GUI for Zori, a Quantum Monte Carlo Program," *Computing in Science & Engineering*, vol. 11, no. 1, pp. 41–47, 2009, DOI: http://dx.doi.org/10.1109/MCSE.2009.5.

[13] J. Hellmers, K. Heiken, E. Foken, J. Thomaschewski, and T. Wriedt, "Customizable web service interface for light scattering simulation programs," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 113, no. 17, pp. 2243–2250, 2012, DOI: http://dx.doi.org/10.1016/j.jqsrt.2012.07.006.

[14] C. Evangelinos, P. Lermusiaux, S. Geiger, R. Chang, and N. Patrikalakis, "Web-enabled configuration and control of legacy codes: An application to ocean modeling," *Ocean Modelling*, vol. 13, no. 3, pp. 197–220, 2006, DOI: http://dx.doi.org/10.1016/j.ocemod.2005.10.002.

[15] S. K. Geiger, "Legacy Computing Markup Language (LCML) and LEGEND–LEGacy Encapsulation for Network Distribution," Master's thesis, Massachusetts Institute of Technology, 2004.

[16] S. Frickenhaus, W. Hiller, and M. Best, "FoSSI: the family of simplified solver interfaces for the rapid development of parallel numerical atmosphere and ocean models," *Ocean Modelling*, vol. 10, no. 1–2, pp. 185–191, 2005, the Second International Workshop on Unstructured Mesh Numerical Modelling of Coastal, Shelf and Ocean Flows. doi: http://dx.doi.org/10.1016/j.ocemod.2004.06.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1463500304000605

[17] P. J. Hurley, W. L. Physick, and A. K. Luhar, "Tapm: a practical approach to prognostic meteorological and air pollution modelling," *Environ. Model. Softw.*, vol. 20, no. 6, pp. 737–752, 2005, doi: http://dx.doi.org/10.1016/j.envsoft.2004.04.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815204001100

[18] Scifes: Scientific front-end systems. https://www.scifes.com/.

[19] I. Gorton, C. Sivaramakrishnan, G. Black, S. White, S. Purohit, C. Lansing, M. Madison, K. Schuchardt, and Y. Liu, "Velo: A knowledge-management framework for modeling and simulation," *Computing in Science Engineering*, vol. 14, no. 2, pp. 12–23, March 2012, DOI: 10.1109/MCSE.2011.116.

[20] I. Gorton, Y. Liu, C. Lansing, T. Elsethagen, and K. Kleese van Dam, "Build less code deliver more science: An experience report on composing scientific environments using component-based and commodity software platforms," in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ser. CBSE '13. New York, NY, USA: ACM, 2013, pp. 159–168, DOI: http://doi.acm.org/10.1145/2465449.2465460.

[21] S. A. Vilkomir, W. T. Swain, and J. H. Poore, "Software input space modeling with constraints among parameters," in *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, vol. 1. IEEE, 2009, pp. 136–141, DOI: http://dx.doi.org/10.1109/COMPSAC.2009.27.

[22] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl, "GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit," *Bioinformatics*, vol. 29, no. 7, pp. 845–854, 2013, DOI: http://dx.doi.org/10.1093/bioinformatics/btt055. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/29/7/845.abstract

[23] GROMACS. http://www.gromacs.org/.

[24] GROMACS manual. http://www.gromacs.org/Documentation/Manual.

[25] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1/.

[26] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. http://www.w3.org/TR/xmlschema11-2/.

[27] M. Bösecke. Pebble. http://www.mitchellbosecke.com/pebble/home. Accessed 12^{th} Feb 2015.

[28] Bootstrap-Themed Tree Widget Documentation. http://jhfrench.github.io/bootstrap-tree/docs/example.html. Accessed 3^{rd} Mar 2015.

[29] TemPSS REST API. https://github.com/london-escience/tempss/blob/master/doc/API.md. Accessed 12^{th} Mar 2015.