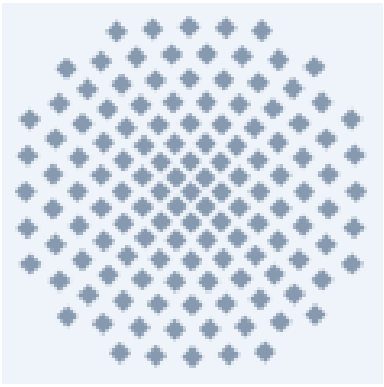

Studienarbeit Nr. 2309

Spar-File Manager
for the Stuttgarter Workflow Machine

Tobias Rohm

Studiengang:	**Informatik**
Prüfer:	**Prof. Dr. Frank Leymann**
Betreuer:	**Dipl. Phys. Dieter H. Roller**
Begonnen am:	**01. Dezember 2010**
Beendet am:	**01. März 2011**
CR-Klassifikation:	**D.2.2, D.2.3, D.2.6, D.3.4, E.5, H2.3, H.4.1**

UNIVERSITÄT STUTTGART
INSTITUT FÜR ARCHITEKTUR VON ANWENDUNGSSYSTEMEN

Contents

1. Introduction

There is no modern business today without the support of computer systems. With the raise of computer networks and the World Wide Web the possibilities and challenges for modern companies have changed.

Computer systems in the past primarily served for the aggregation, manipulation and efficient storage of data. After the broadening of computer networks, the focus changed to technologies supporting data exchange. The possibility to interconnect systems with certain purposes lead to more and more sophisticated forms of computer aided collaborative work.

Coincidental new problems had to be solved. One challenge was to combine the variety of different hardware, operating systems (OS) and software products, using different data types, formats and representations. An early approach, were so called middleware systems to provide a common interface for heterogeneous systems. One well-known specification with several implementations is CORBA (Common Object Request Broker Architecture).

But combining different systems was not the only problem. An evolution of hardware, software and network technologies occurred, with a speed that was formerly not known. The usage of mainframes, computing the demands of many users on a single huge machine, was replaced by interconnected workstations and special servers. Networking became mobile, and the networks advanced. From cluster computing with hundreds or thousands of identical machines working together to solve big tasks, over grid computing to organize heterogeneous systems for collaborative work.

A common term nowadays is the term cloud computing to emphasize the dynamic change in the combination of more and more information- and computing-resources forming, a superior network compound. A future scenario is most likely to happen, ubiquitous computing, which means that network linked information systems will pervade the whole life of human beings.

To the progress of hardware, becoming smaller and more and more efficient, concurrently the software side evolved. With increasing complexity huge monolithical programs became impossible to maintain and handle. Object oriented programming languages encouraged a trend to separate logical software units according to their functionality and assignment. This approach proceeded in higher layers of software modularization. For example the MVC pattern (Model View Controller) emerged, a strict division between data model presentation and the processing of information.

Reusability and data encapsulating aspects lead to component based software architectures. Real implementations were decoupled from their functionality trough interfaces, describing only the operations possible for an object.

But at that point the evolution still carried on. The individual tasks were refined to more and more levels of abstraction: Security aspects, quality of service, maintaining state versus stateless and so on. Stand alone software installations were replaced by application servers running instances of the requested computational services executed in a specialized environment (container). The containers are managing additional aspects like transactional behavior, concurrent access, serialization to stable storage and security.

In between even the great players in software industry recognized the advantage of corporate cross-boarder standards for protocols and description and modeling of data. XML (Extensible Mark-up Language) is such a standard for advanced well defined data exchange and more.

Concerning context, structure, types, parameters and corresponding values of data items XML is able to describe nearly everything. Although XML is not as mighty as a programming language, it is perfectly suited to serve implementations for information descriptions in various areas.

Amongst others, a main meaning for all these efforts was supporting companies to accomplish their individual business goals. The formerly data centric focus on business supporting software was extended by operational aspects. Now the IT system of a company can not only support the work on business information, but also steer, manage and supervise the whole production process. Workflow management systems incorporate a variety of new programming techniques to ease the data processing and exchange within instances of business processes.

The whole course of events and actions could be managed by such a system. The system can invoke the appropriate actions, monitor the progress and react to failures while maintaining the state of a business process. Furthermore a workflow management system is not restricted to one company. Such a system should also be able to use and offer business services to other companies so called b2b scenarios (business to business).

For these purposes the business process has to be modeled and described in an interchangeable well defined format. Early approaches were for example JCL (Job Control Language) or FDL (Flow Description Language) describing the operational aspects of a business process.

Today the de-facto standard among these languages is BPEL (Business Process Execution Language). The possibility to describe business processes of arbitrary complexity demands the support of tools with graphical user interface to develop business process descriptions. There are many sophisticated tools supporting the design of business processes.

Due to a component based approach and the complexity business processes that seem simple at first look, the design can result in an enormous amount of files referring to and depending on each other. To improve structuring and handling, these files should be aggregated in a special superior entity before they are provided to a workflow machine. The files might have to be approved and edited in advance, to ensure that the content of the files goes together correctly, before they can be submitted to the workflow machine.

This Study Thesis presents a tool to collect, edit and package the files necessary to deploy and eventual run actual a business process instances into a ZIP archive, called SPAR file (SWoM Process Archive). The archive combines all files containing the essential information to start and run actual workflow process instances. The archive is intended to work with the Stuttgarter Workflow Machine.

2. Workflow Management

Although the employees of a company might not be aware of, every business inherently possesses some kind of workflow system. Even a traditional porcelain manufacturing consists of many predefined steps to produce the end product. These steps are repeated over and over again. Every participant of the system, maybe only based on experience of the workers, knows in the optimal case at every single point in time what the next action is to be taken.

In the following we will examine a real world example from the author's own experience as a student laborer for a great automobile manufacturer. The example shows in a descriptive way many important aspects of modern production and therefore workflows.

Let's start with a big part of the picture. The factory ground of this company part was housing a number of facilities, all together as big as a town. In fact the plant was a real town, with own fire brigade, police, shops, hotels, railway station and a separate postal code. This huge structure was only responsible for the "end"-manufacturing of the cars. The word end emphasizes the fact that even such an enormous habitat is only handling a subsection of the whole production process, starting with the purchase order for the car until it is delivered to the purchaser.

There is no one knowing all detailed steps necessary to execute the whole process. As well no computer system on this earth is capable to take even one snapshot of the overall status of such a system at a certain point in time. Nevertheless it worked nearly the entire time without noticeable disturbances.

Now we will zoom in. The next zooming level is a big production hall with several assembly lines carrying hundreds of car bodies in an early stage of completion. The lines are moving the bodies tied on steel frames at a slow speed to the next work station. To get back to the big picture, the car bodies, one after the other, made a journey through many areas of the production facility. The actual path was depending on their individual design model and equipment. From a perspective far away they can be seen like they are flowing through the production assembly, forming a structure like the Amazonian river system, with one preliminary final destination, a gigantic parking lot, where the cars are waiting to be delivered over the whole world. On their way they are completed bit by bit to a finished product, a complete automobile. Most of the time, conditioned by the order situation, the flow continued 24 h a day.

The author's job together with his collegians was to insert the headlamps and turn indicators. This was done with an adapted cordless screwdriver and a special caliber by hand. In workflow terms this is called a human task. Amazingly this was done at an early point of completion. One might think this is a step done after most of the car is finished. But at this stage of the production process the car body contained only a small part of the cable loop and the electronic and pneumatic systems. This makes sense. At this stage it is very easy to reach the installation slots for the light components.

This is one property of a good production process. Every single task is placed in the time line at the most appropriate and efficient place. Furthermore other steps have to be accomplished before the next one is possible. In our example the car bodies were already lacquered completely and as mentioned before the supply cables and connectors were laid before too. Therefore it was possible to plug in the control and power supply cables fore the light components.The working steps themselves were always the same, repeated about more than a

hundred times in one shift. But the inserted light components changed corresponding to the purchaser individual desires.

For that reason to every car body was a map of computer printouts attached containing the necessary information. The contents of this map changed from one working station to another. So in parallel to the flow of the product there was a flow of adapted information. Information was not only transmitted in the same direction of the flow. Every time before parts like screws were running low a forklift with fresh supply appeared. The necessary information was transmitted directly to the repository invoking the delivery process.

Furthermore at the end of the every assembly line was a special station for end control. A laptop was connected to the car running a small test program and after successful control transmitting according information to the superior information system. Such an approval is a key component of workflow management systems. Obviously this information flows are one of the key competences for an IT-based workflow management system.

But the system had also to ensure that if something goes wrong appropriate actions were started. Sometimes the end control team was forced to correct failures made on the assembly line before. This meant in the worst case to dismount all components and install the correct ones. This action was somehow similar to in workflow terms a “transactional behavior” called “rollback” and recover to a consistent state. In extreme cases of failure it was necessary to remove a whole car body from the assembly line correct the failures and add it again in the production process. When this was not possible the whole production process had to be started again for this entity.

Failures can decelerate or even stop the whole production process. Problems occurring in front of one workstation were compensated by special puffers holding more than a hundred pre fractured car bodies to be inserted into the flow when the normal supply was not available. Interestingly, most of the time the assembly line stopped completely, were caused by problems with the robots installing the front window after the work station described here. So failures at a later point in the production process can also affect the process progress in front of it.

Congestions in the flow must be avoided with special measurements in advance. But if congestions or other failures happen, the system should be flexible enough to react in an appropriate manner. Disturbances in the production flow should not be underestimated. Most production processes are highly distributed. Every problem or change in a section can have massive impact to the rest and at places difficult to be foreseen.

To complete the picture, some remarks to the human factor. On this individual assembly line it was possible for every worker to define his own speed of work as long as the overall flow was not affected. Eventually that meant the possibility to move up and down the line gaining an additional pause for example. But with this comfortable alternative to break up the monotonously work came an internal order not to be seen by an engineer being too far at the front of the assembly line. This was caused by the fear that this would be a reason to speed up the line movement. A proper workflow management should also take the mentality of human beings into account, as long as they are involved in the business process.

This is also reflected by the phenomenon of so called “Monday Cars”. These cars showed an extraordinary amount of failures passing the quality control unnoticed. The assumption behind is, that the workers are not completely “in the flow” after the weekend. The phenomenon was only revealed because of the, introducing another workflow term, “audit

trail" belonging to the production process. This is another important property of workflow management systems. The possibility to aggregate and analyze information concerning the performance and failures of production processes. This information can be used to optimize the process. An appropriate countermeasure could be for example to strengthen at Mondays the end control.

For rounding the real world example a last story, told by the collegians like a legend. Years before, the gigantic chain moving the assembly line broke. As a result the whole line had to be uncovered to find the breaking point. In the first instance a fluke for the workers being paid for doing nothing was a really costly incident for the company. Not in the first place the charge for the repair. Much more worse was the interception of the production process. Somehow a workflow management system can be seen as this chain. For most of the participating people of a production process invisible, it drives the whole production to its final business goal. Thus a workflow management system must be always available and reliable.

Not mentioned, but also important were security demands. The example should have shown some of the variety of properties and capabilities a workflow management system has to cover. But the main intention was to provide a little insight to the enormous complexity of modern production processes.

After this extensive introduction of a production process in the real world the following sections focus on the computational aspect of workflow management.

The following is mainly based on the book "Production Workflow" from Prof. Dr. Frank Leymann and Dipl. Phys. Dieter H. Roller [LR2000]. For detailed information to concepts and techniques to the subject workflow please refer to this book.

2.1. Workflow Technology

Nowadays nearly every company uses computers to support their business. The comprehension of computer support was and is increasing.

The products of an enterprise are manufactured by a process composed of several units of work. From an internal point of view in many industries the product can be equalized with the business process. For example insurance companies or banks, eventual deal with digital information. That's the reason for coining the term production workflow. In the same way all other enterprises of a certain complexity are dependent on computational support of some kind.

While at first computational support of companies encompassed mainly exchanging, manipulating and archiving data the following generations of workflow systems can also drive operational aspects of an enterprise.

Units of work are done by humans, machines and computers possibly in collaborative work with human beings. The last generations of workflow management systems not only ease the work with information but also can steer the whole production process and even making business decisions.

The advantages are multifaceted. Assuming that there are no hardware and software failures, computers are fast, have an extensible memory and never forget. Properly programmed for special purposes computers are superior to human processing. Even a brief discussion of all workflow aspects would extend this document. Therefore the focus is on the constitutive parts of modern workflow management for the Spar-Manager application.

2.2. Business Process and Workflow

In order to deploy a business process on a workflow management system it has to be modeled in a computer understandable way. Figure 1 (page 10) shows the relationship between the real world and the correspondent computing.

The challenge is to find a suitable open format to reflect all important aspects of the real world business in an extensible and interchangeable model description.

Such a format has to be as abstract as possible to be adaptable to all kinds of business. Furthermore it has to be flexible enough to be prepared for all changes that companies are bound to go through in a globalized world.

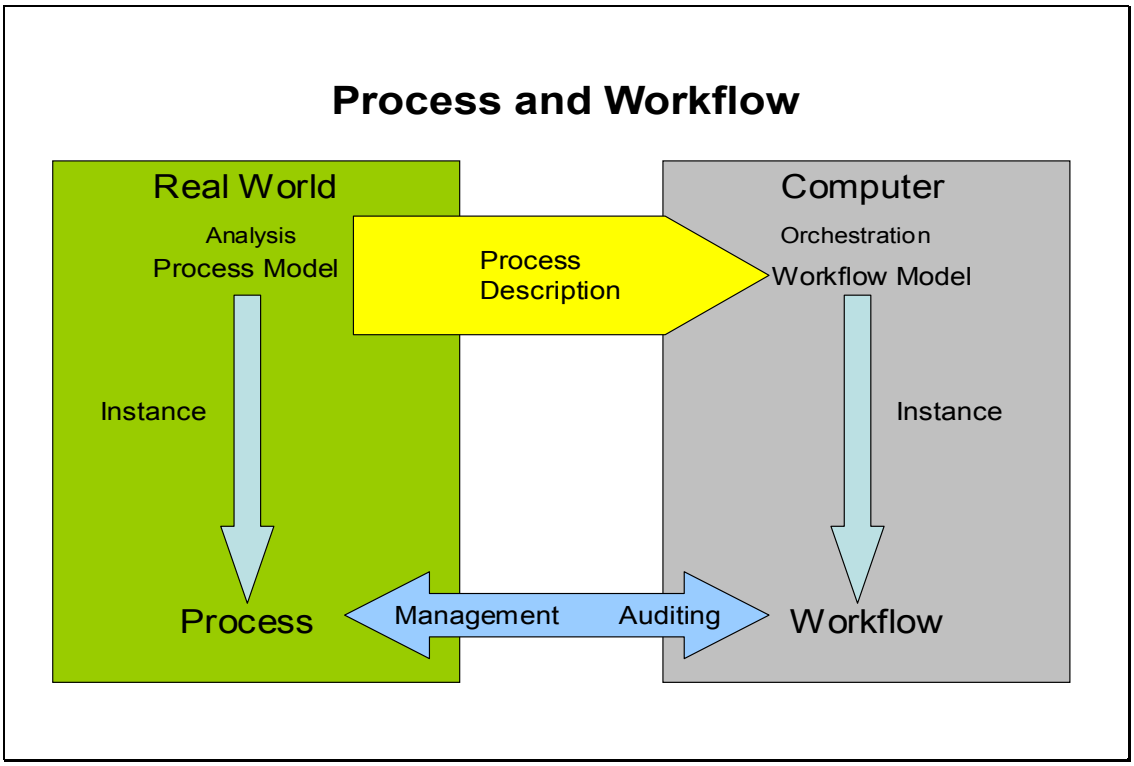

Figure 1: Relationship between Process and Workflow[1]

Due to the complexity of real world business the modeling of business processes has to be subdivided into different fields of functionality. Often this is done by introducing more levels of abstraction. A first important subdivision is between the workflow course and the necessary units of work or the actual activities that have to be accomplished.

While the aim of an enterprise like trading stocks mainly stays the same the actual way to perform activities and the used software may often change.

For this reason the Stuttgarter Workflow Machine follows a two level programming model, which is described in the next section.

[1] The figure is based on figure 1.2 from [LR2000], page 7.

2.3. Programming Model

The flow steered by decisions and realized by invoking a variety of activities is separated from the actual, co called implementation of the tasks that have to be accomplished towards a completed product. The term implementation must not mean running code in a specific programming language.

The task to be done can be everything that satisfies the business demands, even a pure human task, performed without any computer aid at all. Nevertheless a business process driven by a workflow management system needs an interface to invoke activities and receive the results, at least an acknowledgement that the work was done or something went wrong.

A way to do this is to describe an activity as a web service, which is and abstract definition of the functionality of a service. This leads at a coarse grained view to a two level programming model with a part concerning the operational aspects of a workflow called “Programming in the large” and has a so called choreography as an output. The term is used to emphasize that the choreography colloquial determines “what” and “when” (in which sequence) something has to be done. Like the conductor of an orchestra the workflow system leads the single musicians to play successful with each other. Therefore sometimes the term orchestration is used for a process description.

The real implementation or colloquial “how” the work is done, is decoupled from the flow description and covered by the “Programming in the small” part. Hence this document deals with computational aspects, the activities are referenced in the following by web services. As mentioned before, this can be a stand-alone computation for example determining the average of a huge data set or a unit of work performed by an employee using an application at his workstation.

Figure 2 illustrates the correlation with web services managed by an application server. The application behind the web service is decoupled from the actual program, which can be an executable of any kind, for example implemented by Java Beans remotely interacting with a human person, who is in charge for, via Java Server Pages.

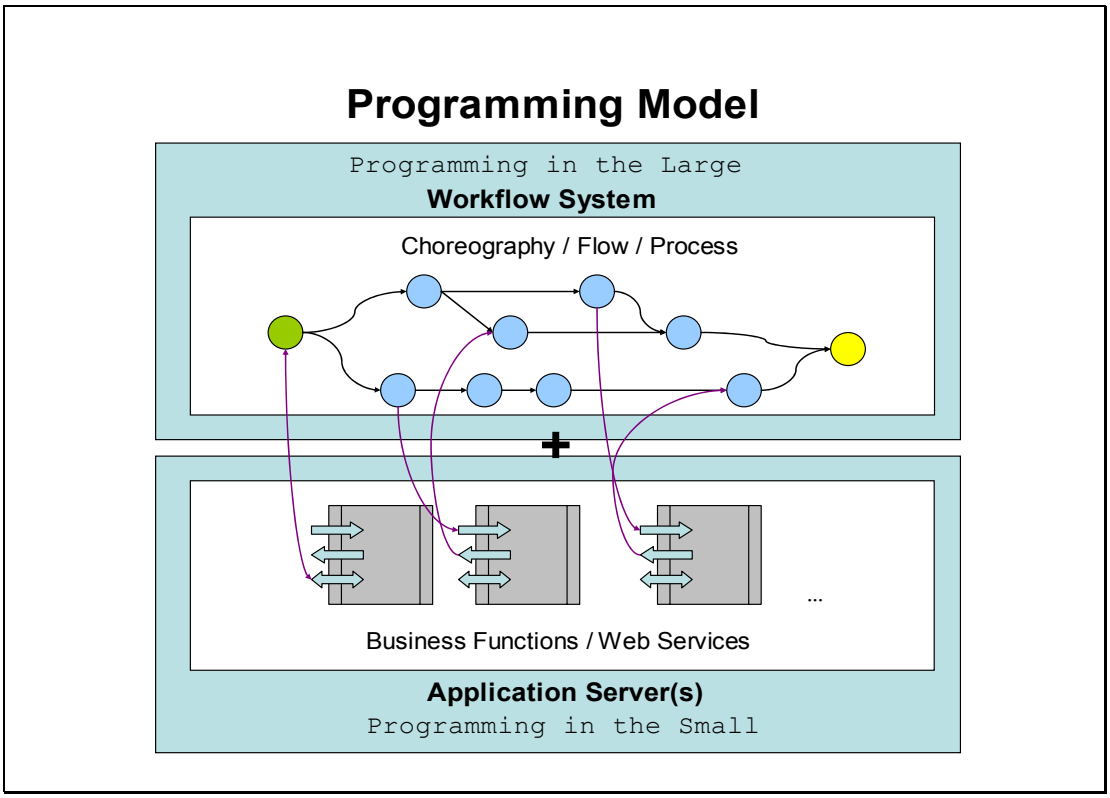

Figure 2: Two Level Programming Model[2]

The choreography of a flow is often represented visually by a directed graph. The actual description of the process model and the participating web services is done in well defined XML files, which are described in the next section.

2.4. Business Process Description and Modelling Files

To handle all the tasks a workflow management system is intended for the workflow is modeled in an abstract manner. That means extendable and suitable for arbitrary aspects of real business companies. As mentioned before a suitable way to do this is via XML. All files related to a business process model for the Stuttgarter Workflow Machine are in XML. Therefore the next section gives a short overview about XML.

2.4.1. Extensible Mark-up Language (XML)

The Extensible Mark-up Language is a meta-language based on text documents for platform and implementation independent data exchange between computers. This is not new. For example HTML is well known and was formerly the main language to provide and exchange documents via the internet. The documents refer by hyperlinks to each other and other resources for example multimedia data. The term mark-up originates from the early printing

[2] The figure is based on slide 10 from the "Workflow Management" lecture WS 04/05, chapter "BPEL-Basics", held by Prof. Dr. Frank Leymann.

industry, where a mark-up in the text advised the typesetters, which letters should be used for the marked text. Today we would say how the text is formatted.

Nowadays the mark-up is done mostly by so called Tags. HTML for example uses tagging to provide information to the interpreting program, for example a web browser, how a text should be displayed.

So why specify another extra language? While HTML mainly deals with structuring and formatting aspects of the text there was a strong demand to extend documents with semantic information. The first generation search engines of the internet were only able to search for key words. With additional context information the search space can be reduced.

A standard example is marking the name of a person with the role for example author the person has in the search context. Of course XML is much more. The intention of XML was to provide an international standard for building text documents that are computer and possibly human understandable in a consistent and like the name says extensible well defined way.

The standard should include structure and context information. The standard was and is defined by the World Wide Web Consortium and tried to respect all scientifically knowledge about language computing.

For theoretic computer science computers just read, manipulate and write data of a well defined language. At the lowest levels these data is encoded in bits. The next level are symbols for example characters. A coherent set of symbols can form a superior entity called for example tokens or, concerning the human speech words. These again can form higher units like sentences.

Hence XML evolved from the information technology sector it is primary dealing with units of data describing data objects. XML introduces data entities, called elements, which are described by nested tagged tokens providing additional information. A set of these entities again can be aggregated in a next level unit, for XML this is a document.

In Order that a document can be understood and interpreted by humans or computers it has to follow specified rules. Formal these rules are defined by a grammar. If a document follows the rules and constraints defined by the grammar, it can be parsed by a program for further computation. To parse a document efficiently the program must be able to rely on these rules.

Basically the document must be syntactical correct. That means, that the document contains only symbols contained in the alphabet of the language and that all words (in correct sequences) belong to the language. Furthermore the words can only appear at correct position in meta-structures like a sentence or a program clause.

All the rules and constraints can be defined in a XML Schema document (*.xsd). The syntactical correctness is a precondition for computing a document programmatically. Otherwise the next step of the program, while parsing, may be undefined. The test for syntactical and structural correctness is called validation.

But validation is only one aspect of information documents. To do something useful with the document the contained information must be interpreted correctly. Formal these are the semantically aspects of a document. Colloquial, the semantics of a document describes what the included information means. The meaning of information depends also on the situation, or more formal the context in which the data is defined and used.

Entities of the real world can be represented by computers in form of objects. XML is closely related to an object oriented way to see the world. So in XML not only the data, structure,

state and name of an object can be described. It is possible also to describe attributes and meta-information for an object.

Since the same token can appear at various locations in a document XML introduces a concept of qualified names and namespaces. Additional to the tagging a name in XML can be associated with a namespace by a preceding abbreviation followed by a colon.

The entities in XML can be complex and nested. Eventually the structure of an XML file forms a tree, which can be traversed by special query languages, for example XPATH.

Listing 1 shows a simple XML file the SPAR-Manager uses to store the path to the folder containing the SPAR Projects.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:SparManagerBaseFolder
        xsi:schemaLocation="http://www.example.org/SparManagerMetaData
        SparManagerMetaData.xsd "
        xmlns:tns=http://www.example.org/SparManagerMetaData
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        C:\Dokumente und Einstellungen\Administrator\Desktop\SPTestFolder
</tns:SparManagerBaseFolder>
```

Listing 1: Simple XML File

The Spar-Manager application is intended to collect, validate, edit and archive the files of a business process description model for the Stuttgarter Workflow Machine. The process is described by different kinds of (XML) files. The next section gives an inside view to the sense and data content of these files and how they are related to each other. The subjections are mainly based on the slides of the lecture “Workflow Management”, WS 2004/2005, held by Prof. Dr. Frank Leymann.

2.4.2. Web Services and WSDL Files

Web Service are service descriptions that shall allow requestors to use the service in an as fare as possible loosely coupled manner. The intention is to hide all details concerning the real service implementation and mainly focus on the functionality the service provides.

The functionality is described in a standardized format by the web service description language (WSDL). A WSDL document is an XML file instance, defined by an XML Schema. The format allows the separation of the abstract description of the functionality offered by the service from the concrete details of the description, for example how (protocols) and where (address) the functionality can be obtained.

From a business point of view a manager is not directly interested in where, how or who performs the work. It is only important that the manager can order and rely on, that the work is done at all.

The WSDL Specification includes amongst other the definition of 6 important elements:

- **type:** Definitions of the data types used for the data exchange between client (user of the web service) and the provider of the service. The types can be defined as XML data types of arbitrary complexity.

- **message:** An abstract description of exchanged data units (messages). Such a data unit can be composed by a set of elements, which are associated with a definition of a data type system, for example XML types (see before).
- **portType (in WSDL 2.0 interface) :** A set of operations supported by the service together with the messages (see before) implicit defining the interaction patter between requestor and provider of the service. These are for example “request-response”: The server (solicit response) or client (request-response) sends a message and receives the answer or one way operations like a notification.

The above elements provide abstract information about the functionality of a web service. Furthermore a WSDL description can contain information about concrete descriptions of the message encoding and end points that provide the requested services.

- **binding:** Specifies the concrete protocol and data format for the operations of a portType.
- **port (in WSDL 2.0 end-point):** Specifies an individual end point identified by a network address (mostly a URI), which is supporting a particular binding.
- **service:** Summarizes a collection of related end points.

Listings 2 to 4 are showing shortened sample excerpts of a test WSDL file of a business process intended for the Stuttgarter Workflow Machine.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
      name="TTProcA"
      xmlns:tns="http://rol.swomTest.org/wsdl/TTProcA/" …
 <wsdl:types>
  <xsd:schema targetNamespace="http://rol.swomTest.org/datatypes"
          xmlns:dt="http://rol.swomTest.org/datatypes">
    <xsd:element name="StringElement" type="xsd:string" />
  </xsd:schema>
</wsdl:types> …
```

Listing 2:Header and Data Type

```
 <wsdl:message name="Message1">
   <wsdl:part element="dt:StringElement"
           name="content1"/>
  <wsdl:part element="dt:StringElement"
               name="content2"/>
 </wsdl:message>
```

Listing 3: Message

```
<wsdl:portType name="TTProcAPT">
  <wsdl:operation name="start">
    <wsdl:input message="tns:Message1"/>
  </wsdl:operation>
</wsdl:portType>
```

Listing 4: PortType

The process defined here is purely abstract. There is no explicit binding to an end point. Explicit bindings are defined in a deployment descriptor file described in section 2.4.4.

2.4.3. Business Process Execution Language (BPEL)

BPEL is a language to specify the behavior of business processes as a web service and related to web services. The usage spectrum of BPEL reaches from abstract processes defining constraints on operations over defining business protocols between business partners up to executable complex business processes between multiple partners.

BPEL evolved from a combination of the graph based language IBM WSFL (web services Flow Language) and the calculus based language Microsoft XLANG. Both of them are in XML and are using web services as basis.

The possibilities BPEL encompasses are very complex to respect all kinds of business aspects. Therefore the following gives only a brief description for important artifacts and their purpose a BPEL document can contain. Some of the artifacts are complemented by examples from a BPEL file intended for the Stuttgarter Workflow Machine (Listings 5 – 8).

- **Partner Links**

Based on a partnerLinkType a partnerLink specifies the static shape of relationships between the process and a partner. PortTypes of web services are referenced as role. A partnerLinkType relates at most two portTypes and specifies the "plugging" of two types of partners or service providers. Multiple partnerLinks can be based on the same partnerLinkType because a process could interact with multiple different providers offering the demanded service.

```
<partnerLink name="TTProcAPL"
             myRole="TTProcA"
             partnerLinkType="tns:TTProcALT"/>
...
```

Listing 5: Partner Link example

- **Variables**

Variables (data containers) contain data that can be stored persistent, for example in a Database. Variables can be used for example to maintain the state of a business process, which may change during the execution and influence the further execution. BPEL variables are scoped and are available by name for each activity associated with the variables scope. The content of variables can be edited by particular assignment activities.

```
<variables>
  <variable messageType="tns:Message1" name="InRequest"/>
  <variable messageType="tns:Message" name="OutRequest1"/>
  …
  <variable messageType="tns:Message1" name="OutInvoke"/>
 </variables>
```

Listing 6: Variables example

- **Correlation Sets**

Correlation sets are lists of properties that are embedded in messages between the web services to correlate the message with business process instances. A property can be a global defined type with a special business semantic. Since a process description can be instantiated multiple times a correlation mechanism is used between partners to identify the accurate process instance.

```
 <correlationSets>
  <correlationSet name="correlation1" properties="tns:correlationProperty"/>
  <correlationSet name="correlation2" properties="tns:correlationProperty"/>
 </correlationSets>
```

Listing 7: Correlation Set example

- **Handlers**

Handlers resemble exceptions from the programming language Java. Handlers define activities that can deal with special situations that may occur within the execution of a business process. These situations are for example faults that occurred requiring the compensation of a long term transaction (similar to a rollback execution for a database).

- **Activities**

Colloquial the activities describe "what shall be done". Since in this programming model every actual activity implementation is described as a web service the activities in a BPEL document are working with web services. There are basic (or atomic) activities and structured (or complex) activities.

Basic Activities:

- **assign:** Manipulate the content of a variable
- **invoke:** Call of a web service (synchronous or asynchronous)
- **receive/reply:** Provide a (synchronous or asynchronous) web service interface
- **throw:** Explicit signaling a failure
- **wait:** Wait for a time interval or point in time
- **empty:** Do nothing

- **Structured Activities:**
  - **sequence:** Execute activities in sequential order
  - **while:** Execute activities while a condition is true
  - **switch:** Conditional execution of activities
  - **flow:** Arbitrary execution of activities controlled by links
  - **pick:** React to external events (messages, point in time, timeout)

```
<flow>
  <links>
    <link name="LinkAB"/>
    <link name="LinkBC"/>
...
    <link name="LinkHI"/>
  </links>
  <receive createInstance="yes"
        name="A"
        operation="start"
        partnerLink="TTProcAPL"
        portType="tns:TTProcAPT"
        variable="InRequest">
    <sources>
      <source linkName="LinkAB"/>
    </sources>
  </receive>
  <assign name="B">
    <targets>
      <target linkName="LinkAB"/>
    </targets>
    <sources>
      <source linkName="LinkBC"/>
      <source linkName="LinkBD"/>
    </sources>
...
     <copy>
      <from variable="InRequest" part="content2" />
      <to variable="OutRequest2" />
    </copy>
  </assign>
...
  <invoke name="C"
        operation="startTTProcB"
        partnerLink="TTProcBPL"
        portType="tns:TTProcBPT"
```

```
          inputVariable="OutRequest1">
      <targets>
        <target linkName="LinkBC"/>
      </targets>
...
      <correlations>
        <correlation initiate ="yes" set="correlation1"/>
      </correlations>
    </invoke>
  </flow>
```

Listing 8: Flow example

A BPEL specification can be found at [BPEL].

The business process description doesn't contain any explicit binding to endpoints, for example a specific network address of a server waiting on requests to a web service. BPEL engines handle the actual execution of business process instances described by BPEL files. To do this the description has to be deployed on the BPEL engine first.

The business process description and especially the binding of partner links are strictly separated from the process description. To run actual instances of a web services or business processes, which are web services on a BPEL engine or application server the engine specific information has to be provided. The necessary data is contained in a deployment descriptor, which is described in the next section.

2.4.4. SWoM Process Deployment Descriptor (SPDD)

In the end a web service, whether it is a partner business process or a web service used by the process has to be executed on real resources. That means using a real processor and files on an existing computer. Web services communicate via messages. These messages have to be created, send and received. At each end must be a process (not a business process, but a CPU process) handling requests and sending replies. For network resources a concrete protocol and address has to be provided.

This is done by the so called binding. Web service addressing specifies endpoint references (ERP) as an information container identifying an end point properly. Endpoint references allow the identification and description of specific service instances. Additional an ERP can be used to generate and customize a service endpoint description dynamically. Furthermore information for an endpoint can be flexibly and dynamically exchanged between services.

The (for endpoint references mandatory) URI of a service can represent a variety of resources, for examples processors or data entities. Therefore an individual end point (instance) must be identified separately. This can be done by reference properties. These properties can be analyzed by a dispatcher running on the receiver side to identify the eventual implementation processing the message content.

```
<PartnerLinks>
  <PartnerLink name="TTProcBPL">
    <PartnerRole>
      <BindingInformation>
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="startTTProcB">
          <soap:operation soapAction="http://rol.swomTest.org/wsdl/TTProcB/startTTProcB" style="document"/>
          <wsdl:input>
            <soap:body use="literal"/>
          </wsdl:input>
        </operation>
      </BindingInformation>
      <ServiceInformation>
        <wsa:EndpointReference
          xmlns:w="http://rol.swomTest.org/wsdl/TTProcB/">
          <wsa:Address>http://DieterRoller-PC.intern:9080/TTProcB/TTProcBTTProcBPTBindingPort</wsa:Address>
          <wsa:PortType>TTProcBPT</wsa:PortType>
          <wsa:ServiceName PortName="TTProcBPTBindingPort">w:TTProcB</wsa:ServiceName>
        </wsa:EndpointReference>
      </ServiceInformation>
    </PartnerRole>
  </PartnerLink>
</PartnerLinks>
```

Listing 9: Binding example

For more information concerning web service addressing see [WSA].

2.4.5. Programming Model, Refinement

The former descriptions lead to a refined, but still coarse grained programming model. Figure 3 illustrates the relationships.

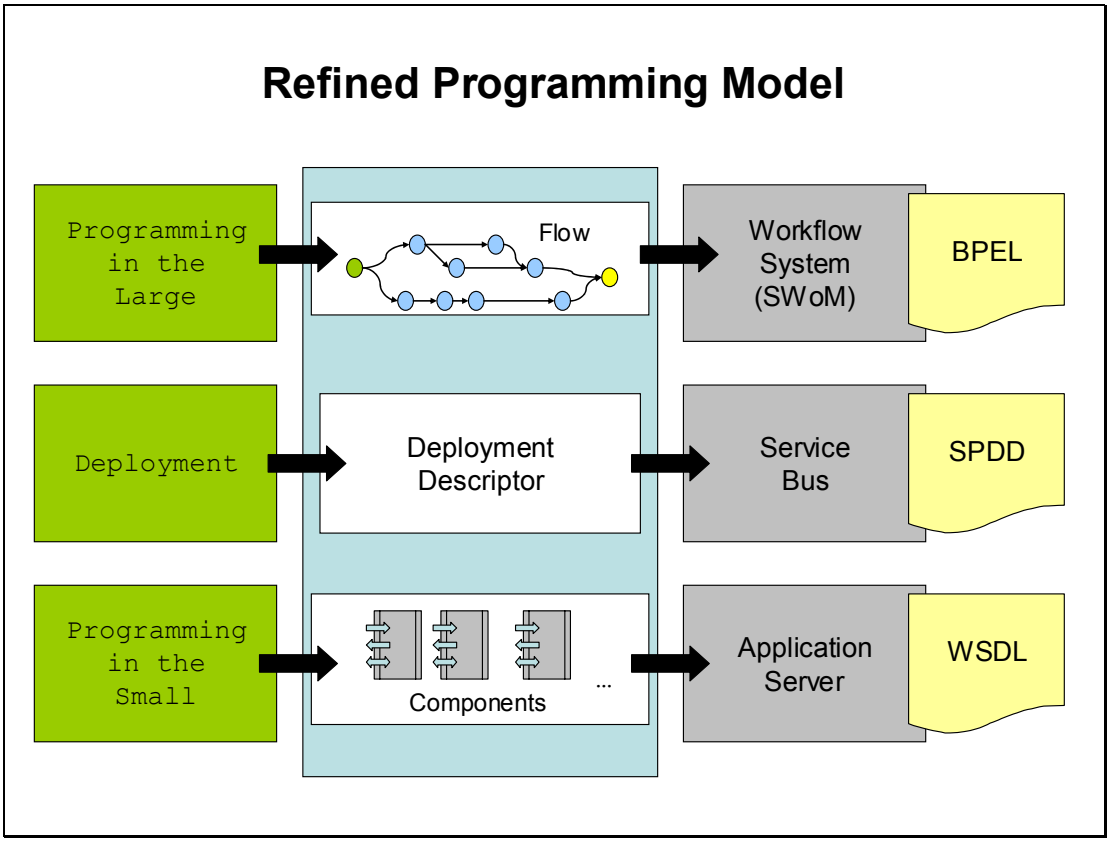

Figure 3: Refined Programming Model[3]

2.5. Stuttgarter Workflow Machine (SWoM)

The Spar-Manager tool is created in the context of a project at the University Stuttgart to develop a workflow management system, the Stuttgarter Workflow Machine.

Workflow management systems can follow a variety of architectures. The first approaches were mainly centralized. Further approaches advanced and became distributed and included support for mobile resources. First prototypes with peer to peer architectures emerged. All these architectures have advantages and disadvantages.

The aim of the SWoM project is to affiliate all advantages from the variety of system architectures and to exploit the actual knowledge about workflow systems. Eventually this covers all available modern IT technologies.The Stuttgarter Workflow Machine implements (partly) the WS-BPEL standard for business process descriptions and follows the two level programming model introduced in section 2.3.

In the end the Stuttgarter Workflow Machine will be a sophisticated, state of the art workflow management system. For more information see [SW2011].

[3] The figure is based on slide 14 from the „Workflow Management“ lecture WS 04/05, chapter “BPEL-Basics”, held by Prof. Dr. Frank Leymann.

3. SPAR-Manager Application

The following part describes the considerations and decisions necessary before starting the real implementation. First, the requirements describing the functionality and operations the tool must and could provide are, presented. The section 3 is structured according to the complexity of possible usage scenarios and the correspondent actions and presentations. Starting with indispensable basic requirements the following subsections describe a set of useful optional functionality options.

3.1. Tool Requirements

The section is subdivided into three parts describing the functionality the SPAR-Manager tool could include. It starts with the absolute indispensable functions in the subsection 3.1.1**.**

Useful functions, but in the first line not an absolute must for the tool are covered by the section 3.1.2.

The subsection 3.1.3 gives an outlook on continuative thinkable functionality a sophisticated tool could encompass.

The ad-joint functionality and corresponding usage scenarios described here, are the determining factor for the implementation design decisions in the following sections. Of course there is no claim for completeness.

In order to that there is a first naturally requirement for the design: The tool should be implemented in a meaningful modularization, ensuring that the code can be easily extended or parts (components) can be replaced without affecting the whole program.

3.1.1. Basic Requirements

In the first line, the tool should support and help users and developers of the Stuttgarter Workflow Machine to compose so called SPAR files (SWoM Process Archive). Such an archive must contain all necessary files to run an actual business process instance on the workflow machine. The embedded files in a SPAR file archive describe a business process model. After the archive was successful deployed to the work flow management system the process model is accessible. Equipped with the contained data the workflow machine can instantiate and run actual instances of a business process.

This means the archive includes a combination of (at least) one process deployment descriptor file (with suffix *.spdd) and (at least) one business process execution language file (with suffix *.bpel). Additional the archive must contain (at least) the web service description file (with suffix *.wsdl) of the process. These files correspond to the “Programming in the Large” part of the two level programming model described in the section 2.1.3.

Furthermore the archive will encompass zero or up to theoretically an infinite number of other web service description files. These files contain the description of the attended web services related to the process. They provide the necessary information for the “Programming in the Small” part of the model. Due to their different role in the programming model the files should not be mixed up. Minimal these WSDL files should be aggregated in a separate folder (for example with the name WSDL).

For manageability, and because all files in conjunction describe an individual business process the files shall be packed into a common archive, the SPAR file.

The given archive format is the well known ZIP. Acceptably ZIP is ubiquitous in the Java world (the corresponding classes are already implemented in the package `java.util.zip.*`) and in Eclipse, the IDE that will be used to implement the tool.

Although the name ZIP was chosen for the algorithm to imply speed, packaging and compressed encoding takes time. But since the tool is user driven and the encoding is the finishing step of the user interaction timing constraints are not relevant here.

The extracting time at the side of the workflow machine, in the face of the complexity of the operations performed to start a business process instance, will not attract attention at all.

More important is, that ZIP encoded archives are able to save disk space efficiently. XML is verbose, and since the files are mostly generated automatically, XML files can become very comprehensive.

Even more important is that ZIP archives can not only preserve the complete directory and file structure, but also attributive meta-information like the time and date of the last access. For additional functionality (see the next sections) this may play a major role for extended versions of the tool.

Since particularly the WSDL files can contribute to more than one business process and therefore may be changed in the context of another business process they cannot be simply referenced in the file system. Anyway they must be encoded in the archive, which means the creation of a copy.

Copies of resources even performed on a local file system, inevitable produce consistency problems. What about a SPAR file, that embodies files that will be changed at a later point in time? Even worse the name web service implies that the services and also their description can be hosted on remote sites. The implications of a business process funding on a distributed system are surely out of the scope of this thesis, but should not be ignored completely.

For the fundamental functionality the subject consistency is left out. All origin files are assumed as static final resources.

Separating the former problems leads to an at least project based approach, which means physical copies of the contributing files are aggregated in an individual structure called SPAR project. The origin files will be used for reading only and therefore not changed. Furthermore the SPAR project will be the primary entry point to an adequate composition of the business process crucial files.

The primary task of the tool is to select and combine the “appropriate” files in an archive. For pure archiving purpose no extra tool would be needed. One of the many ZIP tools available could be used. Therefore the first additional functionality of the SPAR-Manager is to disburden the user from creating proper file structure. Since the XML files are related to each other selecting the appropriate ones, based on file names only, would be bothersome.

Amongst others for this reason XML files can be provided in a human readable form. Expectedly, to pick an appropriate file, the data content of the file has to be examined. Therefore an appropriate tool should provide at least the possibility to display the files. Helpful too, would be the possibility to display more than one file concurrently, to justify if their related entries fit to each other.

From all this considerations some main conclusions are following:

- Obviously the tool should provide an intuitive, guiding graphical user interface.

- The tool should provide comfortable file browsing functionality.
- The structure of an appropriate archive should be predetermined and separated from other archives (SPAR project).
- The structure should be presented in a descriptive way.
- The tool should be able to display the content of the participating files, supporting human perception.

3.1.2. Additional Functionality

The final products of the SPAR-Manager are archives intended to be submitted to the Stuttgarter Workflow Machine. Obviously this task can be eased by advancing the SPAR-Manager by the functionality to do this directly out of the tool. A “Submit to SWoM” button would be a nice extension for the tool, hiding the necessary operations to provide the archive to the workflow machine.

For that purpose additional functionality and menus to define the connection to SWoM would be required. Once the connection is successful established the workflow machine could automatically try to start up and run a business process instance depending on the information in the archive. The definition of the interconnection further depends on way the SPAR file is submitted to the workflow machine.

What seems to be “nice to have” at the first look adapts not to every usage scenario. To design business process descriptions or for testing a workflow management system a feature automatically starting business processes would be convenient. On the other hand such a feature can be extremely dangerous utilizing the system in real business. Users for example, tend to try buttons just because they are present. This is not bad in principle, but as discussed later a proper user interface should prevent users to start a business process instance accidentally.

Therefore the first step to submit the archive to SWoM should be only to copy the file in a special directory, which SWoM uses to access the archive. Additional functionality should be offered extended by accordant request prompts and warnings.

Due to the fact, that business process description can have nearly arbitrary complexity starting up a business process execution is an extensive task. Even under the assumption that all needed and appropriate files have been selected, it would be very annoying if a business process instance crashes at startup or even worse amide a running instance because the description files are faulty.

To exclude at least structural and syntactical errors in the participating files it would be useful for the user to have the possibility to check the correctness of these files in advance.

XML provides a convenient way to do this. XML Schema files (with suffix *.xsd) and Data Type Definition files (with suffix *.dtd) can not only serve as templates to create actual XML documents instances. Moreover existing XML documents can be verified against the definition files, whether they conform to the structure, data types, tagging and namespaces defined in the template, or not. Such a Validation encompasses not the entire semantically correctness of a XML document. A successful validation doesn’t guarantee that the actual data entities, for example parameter values, are meaningful for the according business process descriptions. Nevertheless validation can prohibit many possible sources of errors.

Of course the possibility to validate the member files requires additional implementation and resources overhead. The corresponding Schema files have to be provided and both, the XML document and the Schema file must be parsed.

The capability to detect errors does not make sense without additional functionality to outline and correct errors. Thus the display windows should be enriched for editing functionality. Again a summary of the last considerations:

- The tool could be extended by a “Submit to SWoM” button, equipped with the necessary functionality
- The tool could be extended by a “Validate” button, equipped with the necessary functionality.
- The tool could be extended with user friendly menus to configure the connection to SWoM.
- The tool could be extended with an integrated data model regarding the relationship between the XML workflow documents and their definition files.

3.1.3. Extended Functionality

In the former sections two dimension of work supported by the tool emerged. The first is finding, managing and aggregating the proper resource files for the SPAR archive. The second is the inspection and editing of the files. These two dimension lead directly to possible extension for the user supporting capabilities of the SPAR-Manager application.

Following the first an extended resource management would be thinkable. What happens when resource files are revised? A simple step could be to inform the user, when resource files, contributing to the SPAR project he is working on, changed since the last archiving and let him decide whether the file should be updated or not.

But imagine only the IP-Port of a web service should be changed. This means that every business process description working with the web services has to be found and updated in SPAR-Manager by hand. In a real world workflow system such effort is not acceptable. Additional functionality could for example provide a function to check all projects if they are still synchronous with their origin files and offer adequate actions.

A next step could be, to add an additional abstraction layer to the resource files management. Instead of direct access to physical files and absolute paths a uniform resource identifier (URI) could be used. The last line, printed in bold in listing 10, which shows an excerpt of a sample WSDL file, defines a namespace using an URI (the part enclosed in quotation marks).

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="TTProcA"
        expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
        suppressJoinFailure="yes"
        targetNamespace="http://rol.swomTest.org/TTProcA"
        xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
        xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
        xmlns:tns="http://rol.swomTest.org/wsdl/TTProcA/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Listing 10: XML Header Example

This is a common praxis to ensure the uniqueness of namespaces. Furthermore such an URI can be a valid web address leading directly to the website hosting information to standard namespaces. In this case this is the corresponding site to XML Schema. Adding the file suffix (*.xsd) for XML Schema forms a uniform resource locator (URL) to the XML Schema file that is defining XML Schema files.

This approach has the advantage, that the resource can be maintained in a centralized manner, ensuring that the rest of the world has a consistent access to the resource. This pattern could be utilized to provide the same resource to different business process descriptions. Instead of downloading or collecting the necessary files from the local file system for a SPAR project, it would be possible to load them dynamically, only when needed. This would also ensure that all SPAR project rely on the same resource.

What seems to be a nice solution for many problems concerning the finding and updating of files at the first look, encloses some dangers. Managing concurrent access to files or other resources is not an easy task. In informatics this is related to the term “consistency”. Colloquial the challenge of consistency is to provide any process accessing resources the correct ones he needs at every point in time. The “basic” design of the SPAR-Manager tool provides a so called “weak consistency” model named “read your writes”. Since the manager accesses just copies of the resource files many conflicts can be avoided.

Considering business process instances the situation looks a bit different. Allowing the workflow machine to access SPAR files dynamically, while process instances are active, is an example for the problem. What happens when the SPAR-Manager tool changes files at the same time? In the worst case the change could make a formerly stable running business process unable to complete. The problem is that the SPAR file contains information controlling the course of a business process instance dynamically.

Before seriously using the tool in a business critical section, it must be ensured that the operations of the tool do not affect any other parts of the workflow machine. A possible solution is simply prohibiting the access to SPAR projects actual submitted to SWoM at the manager.

In the end, these considerations may demand to add an additional tier. For example, adding a connection to database managing the mapping and access to resource files. At least it shows, due to the complexity of workflow management, even without a distributed and concurrent file access, a comparatively simple tool like SPAR-Manager has to be treated and designed very carefully.

Managing consistency definitely exceeds the circumference of this study thesis. But it is a meaningful starting point for further work. For a closer look at the subject consistency in distributed systems, and a sophisticated workflow management system is inevitably confronted with problems of distributed systems, one of the comprehensive books related to network technologies of Tanenbaum, Andrew S. shall be recommended [TS2007].

The second dimension of working with the SPAR-Manager tool concerns the editing functions to prepare SPAR file archives to be submitted to SWoM. The files populating a SPAR file archive do have an origin. Many of them can be downloaded, especially the XML Schema and WSDL files. But all of them are the result of a creation process, mostly supported by tools that support automatic generation. Particularly the files describing business processes (BPEL) are constructed with aid of other tools integrated in a development environment for modeling business process descriptions.

Primarily the tool should be integrated in the business process modeling framework. Finally a sophisticated workflow management system will provide the developing process or orchestration of a specific workflow model description as a workflow. Like a XML Schema is specifying how a well formed XML Schema has to be, the system self will steer the development process of its own input.

So because the SPAR file generation is only an intermediate step in the business process developing process it should be possible to be used directly or invoked by the workflow machine at the right stage in the framework. BPEL files for example are created with the help of graphical design tools. The symbols and notations used to orchestrate a model are actually specified as a standard. For more information there is for example the book BPMN 2.0 [AT2009]. Links towards many other books or tools concerning BPMN can be found at the web site [OMG]. In the end the SPAR-Manager application should be integrated seamlessly between the modeling step and the submission to the workflow machine, with as few as possible additional user interactions.

The last two sections are describing functionality that is not straight in the tasks assigned to this study thesis.

3.1.4. Standard User Interaction

Returning to the fundamental functionality Figure 4 shows a flow diagram visualizing the basic user interaction: Selecting files and add them to a SPAR file. The flow begins at the upper left corner and ends at the right upper corner. Since a SPAR file embodies more than one file, the steps between import and save will be repeated several times in a realistic usage scenario. This is indicated by the “extra” flow in the grey box at the lower left corner.

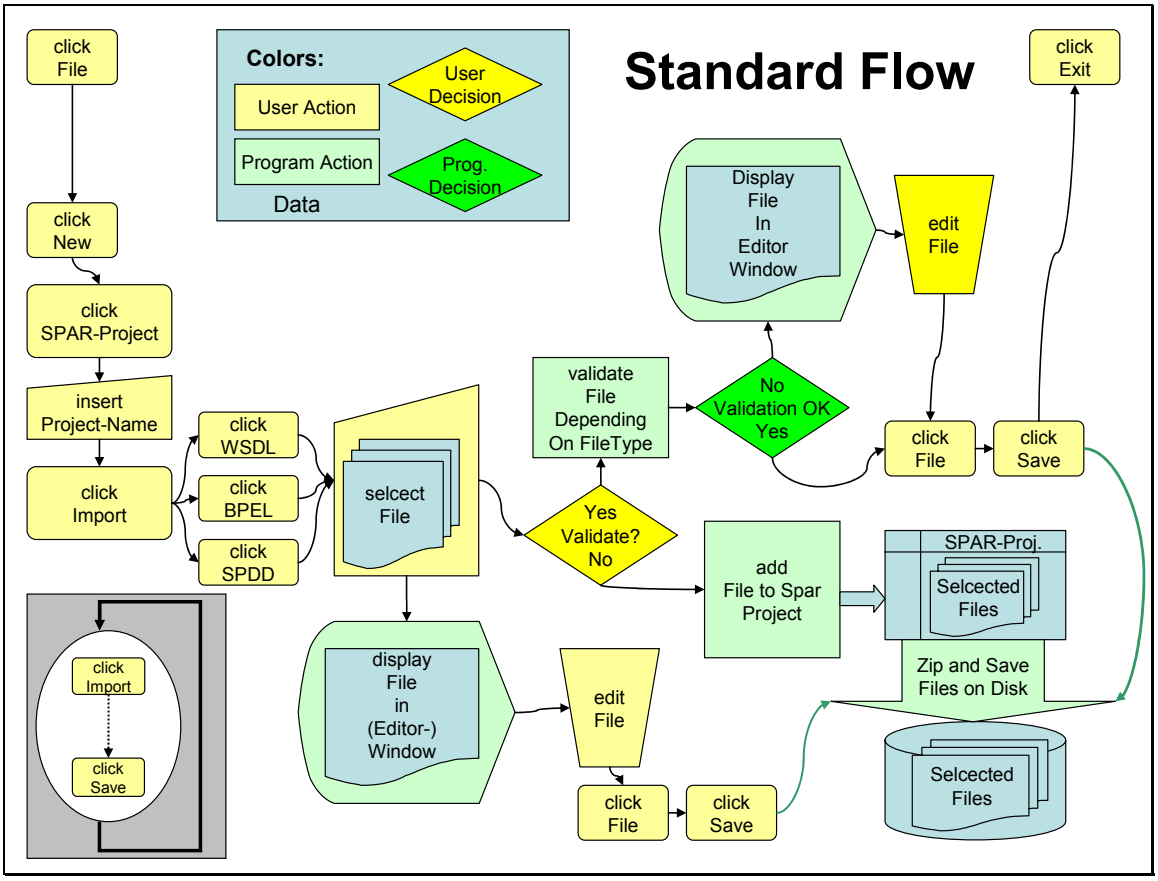

Figure 4: Standard User Interaction Diagram

After elementary considerations about the functionality, this is a good point to remember the "eight rules of design" from Ben Shneiderman, before designing the graphical user interface. This will be done in the following section.

3.2. Graphical User Interface Design

The eight rules presented now are not a strict citation. They are rather used as control points and interpreted directly in relation to the intended application, the SPAR-Manager tool. For the exact source and elaboration see
[SP2010].

Each rule follows a brief explanation and marked with an arrow (➔), the consequences for the SPAR-Manager Tool.

(1) Strive for consistency.

In this context, consistency corresponds not to the data files. Here it aims to single user actions and sequences of user actions. In similar situations she same actions should be required. Furthermore the terminology, especially naming and labeling, should be the same throughout all menus, prompts and screens. That applies to commands too.

➔ The explanations imply to control all signifiers to be the same when they belong to the same resource. Concerning the actions and commands an example: If the user right clicks on a

resource file the same sequence of actions offered (commands) should be invoked, as if the user has used the “File” menu entry from the `ActionBar`.

(2) Enable frequent users to use shortcuts.

Shortcuts are mostly offered in two ways. First there are Hot-Keys. They are often a combination of two or more keys pressed on the keyboard simultaneously (e.g. Ctrl-Shift-Alt) or the F-Keys, invoking actions without navigating through menus. The Second way are icons in the so called `ActionCoolBar`, doing the same when clicked. Both of them avoid annoying navigation through menus with submenus and so on. With the users experience about the system increasing, the desire to pace the speed of interaction is growing. Therefore abbreviations and Hot-Keys can be very helpful for an expert user.

➔ Since shortcuts are an additional feature, they should just be added to the graphical user interface of the tool.

(3) Offer informative feedback.

For every action the user performs the system should provide an adequate feedback. For minor or frequent actions the reaction can be modest. Major and seldom performed actions should provoke a more substantial reaction.

➔ Major actions of the SPAR-Manager will for example be the import of new files, or the creation of a SPAR archive. The import could be accompanied by the simultaneously opening of the file in an editor window and the marking of the file in the archive outline, if necessary expanding the tree viewer to the corresponding position.

A SPAR archive creation could end up in an extra popup window showing an overview of the content and status (validated) of the created file.

➔ Minor actions would for example be the saving of changed member files. The completion of this action could be indicated by disabling (not hiding completely) the corresponding action entries in the menus and the related icons in the `ActionCoolBar`. This would show the user that the action cannot be done at the moment because an additional save without changes is absolutely not meaningful any more. Accordingly the action should be enabled again after changes to files are made. This applies for the validation of files too. Additionally an explicit commendation, indicating that the file is already validated, might be useful.

(4) Design dialog to yield closure

Sequences of user actions should be grouped and follow a meaningful structure. They should provide an identifiable beginning and end enclosing the subsequent actions to perform the overall task. The informative feedback from the last rule should provide a sense of relief to the user, having successfully accomplished the task. Then the user can free his mind from optional action flows, and contingency plans. Furthermore it indicates that the user could launch in the next piece of work.

➔ Correspondent to section 3.1 two subsequent flows of work can be identified. The first one starts with the creation of a new SPAR project. The middle part consists of several steps

to populate the project with the appropriate files. After the work is done the project should be saved and maybe an intermediate archive is created.

The system feedback could include a notification that the files of the project are not all validated yet.

This would leads to the next major task, validating and if necessary editing the files until they are all validated. In the end a validated archive is created and can be submitted to SWoM as a final step, which is actual the start to a new process.

A good system should guide the user intuitively into proper action sequences like the ones described here.

(5) Offer simple error handling.

First of all, users should be prevented from actions that can lead to errors. However errors cannot be imposed completely. When errors occur, the system should react with adequate error feedback, and furthermore offer simple and comprehensive mechanisms to handle the error.

Minimal an error message should contain hints why the error occurred. The author himself will never forget the error message of a graphic design program ending up with the text: "…, "bad things will happen!". Unfortunately the announcement was true enough.

Error messages like "An unknown software exception occurred" prove only that the application was not ready to be dispatched and are making users furious.

Again for the SPAR-Manager two categories of errors are presumable.

➔ The first category has to deal with errors concerning files, while composing the SPAR project. An example for error prevention would be not offering the possibility to import other than WSDL files into the WSDL folder. While the tool might not search for them at other positions, the user has the opinion it is already added properly to the project. Other error handling is obvious, like feedback of the form: "The file does not exist!" "Should it be created?"

➔ The second category of work will be confronted with errors in the files themselves. For this purpose the system should indicate lines of XML text, supposed to be erroneous, with red icons in front and underline the suspected part of the line. Additional, adequate correction actions could be offered if the user clicks with the right mouse button on suspected text parts.

Automatic code, or in this context XML content correction, is not an easy task. Because the member files of a SPAR project depend on each other and even worse XML files can reference other XML files, it is anything but easy to determine the origin of successive errors. It can easily happen that parts of code are interpreted as incorrect even though the error offspring lies in another file. To that effect offering appropriate correction actions is a mixed blessing. Such corrective provisions can even lead to a vast amount of new errors. Then at least the user's attention should be called, that the action might be dangerous.

(6) Permit easy reversal of actions

The last problem for example is a good reason for this rule. If an error correction action just leads to even more errors, the user will have a strong desire to return to the state of work

before he performed the action. Furthermore sometimes users are just missing or confounding action components of a graphical user interfaces. For this cases too, an “undo” function would be useful.

Again two dimensions of actions to revert can be determined. But somehow the complexity of the problematic is reversed.

The undo of editing actions like typing in texts is well known, and can therefore be performed comparatively easy. This is simply done by keeping track of the changes and the position where they were made. Modern editors can revert about hundred editing steps and more. But like in mathematics some functions cannot be unambiguously or inverted at all. How is the creation of a SPAR archive “un-done”? Just delete it? Deleting anyway is an interesting matter. If the user deletes a member file from a SPAR project, how is that action undone? Just copy the file back would, maybe parochial, mean to hold an untouched reserve copy because the original file could have been changed in the meantime. Besides the storage expense, to state precisely, the fact that a copy exists denotes that the file was not deleted. The complex of problems is getting even worse trying to provide the opportunity to revert chains of such actions.

➔ For the SPAR-Manager tool the discussion above means that amongst others a file is not “deleted” from a SPAR project but rather “removed”, although the undo demands then an additional layer in the file model in between the SPAR project and the actual files.

Projects in contrast can be deleted, but than the whole information and work performed on the project is irretrievably lost. For a start, the possibility to undo some actions should not be provided at all, and the user should be reminded that this individual action cannot be undone.

When undo functions are provided for complex operations they should be revisited and handled with care.

➔ As mentioned before the usual reversal actions for editing operations should be provided, like the undo of copy and paste insert actions or deleting of text passages.

Other actions for example the validating of a file cannot be undone meaningful, because the withdrawal makes no sense. A file is valid or not. Reverting “higher functions” like “submitting to SWoM” would require intervention in the domains of other program parts of the workflow system. The withdrawal of a SPAR archive could have massive impact to the ability to work of the workflow machine. Therefore actions like that must be forbidden generally. After the submission to SWoM the copy of the SPAR file for SWoM leaves the area of responsibility of the SPAR-Manager for good.

(7) Support internal locus of control

Users strongly desire the sense that they are controlling the system and not vice versa. So the system should not force the user into a respondent role. At every point of the possible action chains the user should feel like he is the leader of the process.

➔ This rule is somehow straightforward for the SPAR-Manager. Since the manager is a tool and tools are implicit rely on to be used by someone. So in the context of this application the rule implies mainly to respect a polite conversational style interacting with the user.

➔ More important is the rule for another design decision. Temporary there was the consideration that the user doesn’t need assertive information what is done with the member

files of a SPAR project to submit them to the workflow machine. So the complete part of archiving the member files could possibly be hidden from the user. But in respect to the rule and the fact that the estimated users for the tool will be experts, wanting to know what they are doing, the preferred decision is to leave the archiving process under control of the user.

(8) Reduce short term memory load

Memory here means not computer memory like random access memory (RAM). It refers to the memory of a human brain. Although a very interesting subject, the mysteries of the human brain are of such complexity that eventually every IT problem looks pale in a direct comparison. But there is a public assumption that the human memory can be divided in two parts. These are “short term memory” and “long time memory”. Sometimes this is compared with the fast main memory and slow stable storage provided by for example hard disks. The comparison is extremely inaccurate and it is a good question, which of the imaginations influenced the other.

But as a matter of fact the capability of a human being to remember visual input, presented only for a short time, is bounded. This applies also to graphical user interfaces. Surely the exhaustion of the user’s mental capabilities by the interface should be minimized to leave capacity for more important tasks.

Therefore the workspace should be designed as simple as possible. Immense changes at the same point in time should be avoided. Multiple page displays should be consolidated and window motion time be reduced as most as possible.

For the SPAR-Manager the rule implies, again following the two main task dimensions, to subdivide the main workspace into two separated subordinate workspaces. The first one should present the selection and combining part of the SPAR projects. The second one deals with the editing and validating functionality for the individual files.

➔ The SPAR project structure and the display of member files should be presented in an overview, presenting the actual projects and their individual content of files. The overview should be easily adaptable to the display of files shown in the second subsidiary workspace maintaining the file data content. Many users are already used to a customizable tree view of file structures. Therefore a tree viewer is a proper choice.

➔ The display, editing, and validation part of the SPAR project should show the data content of the member files in a consistent way. It should be possible to display several files at the same time in the same section of the workspace, overlapping but structured. For example this is practicable with the well known record card style enabling navigation through the files over the registry.

Especially the last rule emphasizes an obvious conclusion. The graphical user interface of a tool should be kept as simple as possible. Simple, means not to omit meaningful features or display components. Because amongst others, supported by tools to develop user interfaces, it is relatively easy to add additional features. Many applications today provide a variety of function buttons and gimmicks.

May be this is sometimes necessary to present an application as professional. Nevertheless the SPAR-Manager is intended to be a tool. Like professional hand craft tools it shall not amuse

the user. It should be, if possible unbreakable, and furthermore just serve the user in the most effective way to do his job.

3.3. Mock-Up

A proper initial design step is to create an abstract picture of the possible appearance of an application without any functionality, the so called mock-up. Figure 5 shows a possible appearance of the SPAR-Manager graphical user interface collectively.

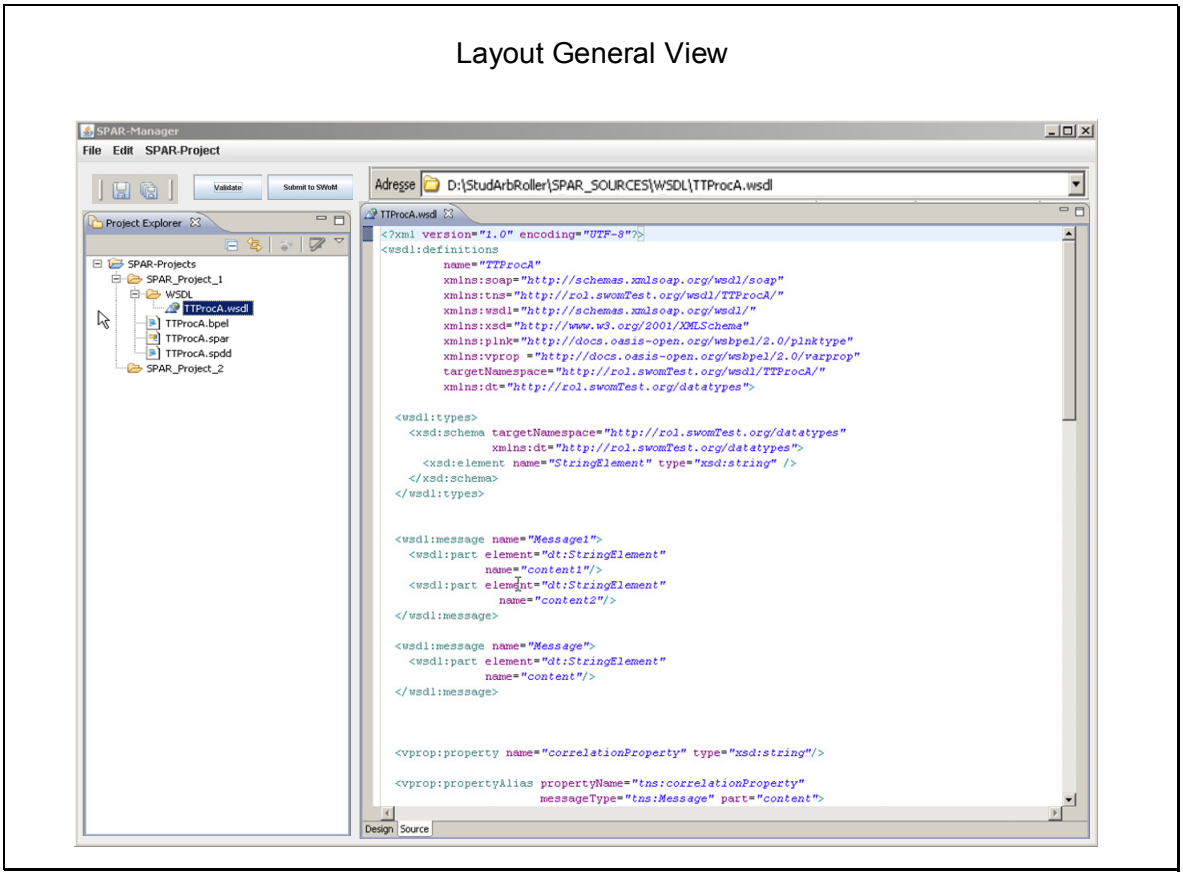

Figure 5: Preliminary Mock-up

Accordant to the precedent considerations, the workspace is divided into two sections. The left one presents a SPAR project and the embodied content respectively the content structure and member files as a tree. A WSDL file is selected. Aside on the right is a section for the editing purposes, showing a sample WSDL file in an editor window. The registry entry at the top determines the active file. The entries at the bottom lead to different views to the active file data content presented as text. Only implied are possible short cut icons in the `ActionToolBar` to save files or all files, and buttons for major actions for validating and submitting the result of SPAR projects to SWoM. The `ActionToolBar` entries are following the accustomed kind, established by Microsoft Windows, which has become comprehensive. The "Address" component, presenting an absolute path to a resource file, shall show the origin where a member file is located. Of course the layout of the mock-up serves only as an initial design for progressive intentions.

The next picture shows in outlines the next detail level concerning the content op popup menus and their relatedness to user mouse clicks. Somebody who has already gained

experience with the Eclipse development platform will recognize the affinity. This is not entire unintentional. In fact, both mock-up pictures are a combination of screenshots origin from a fake Java application and an adapted eclipse project.

Not only because Eclipse is the authors favorite IDE many of the GUI concepts like the tree view of resources, a separated editing window and an project centric way for SPAR file production, are well approved and tested approaches. They are a proper choice for the requirements for the SPAR-Manager requirements described extensively in this section.

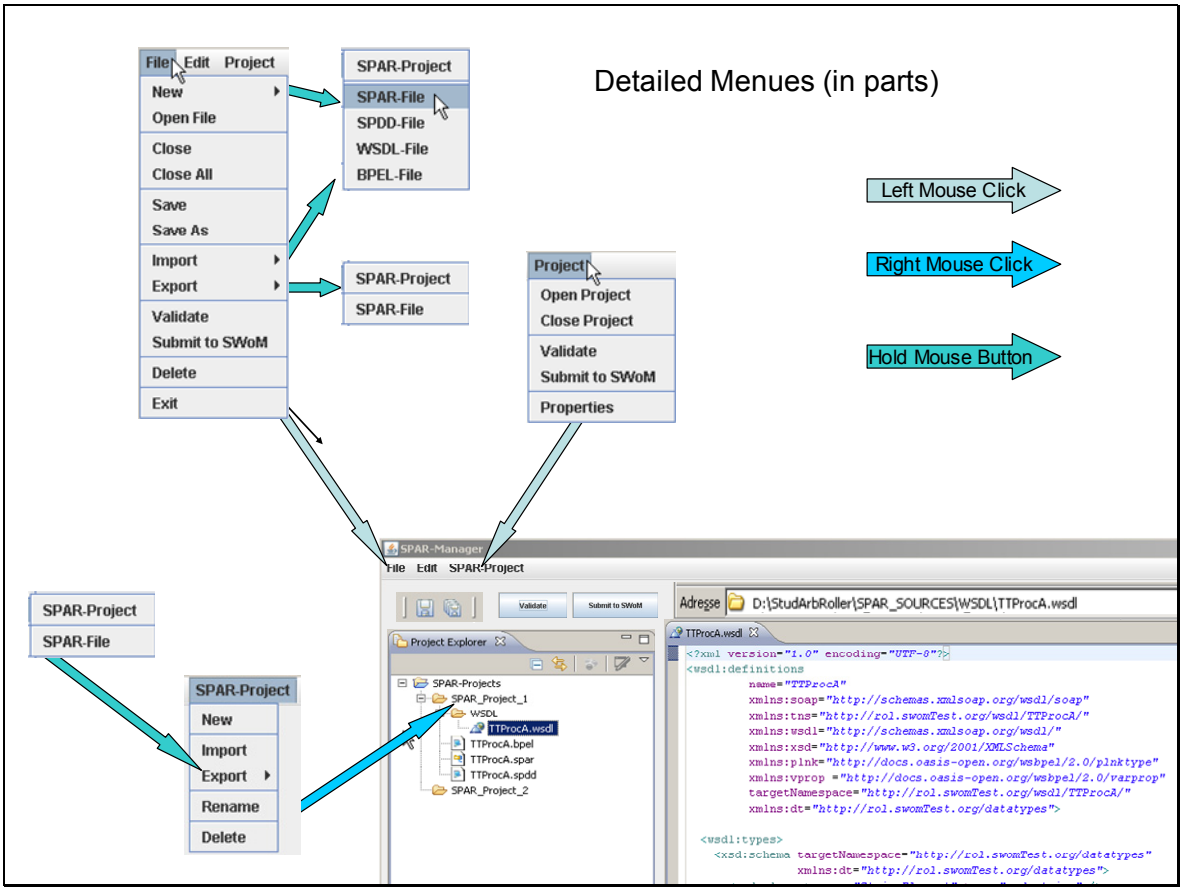

Figure 6: First level Menus and User Actions

The lower left corner shows two example menus. The arrows and their explanation are indicating the user actions. For this picture a blue arrow means that the user clicks with the right mouse button at a SPAR project. Holding the button the user can navigate through a pop -up menu offering the corresponding actions that can be done with a SPAR project. In this case amongst others, the user could export the SPAR project as a project, which can be used for further work on the file assembly, alternatively as a SPAR file archive, which can be submitted to SWoM. Of course simultaneously the blue marking of the “TTProcA.wsdl” file indicating that it is currently selected, and therefore shown in ay extra editor window, would migrate to the SPAR project label.

After clarifying the requirements for the tool, and the resulting considerations for the graphical user interface leading to an initial mock-up, the next sections describe the steps to the realization as a concrete implementation.

3.4. Implementation Design

Consecutively, behind the graphical user interface must be a practical implementation, performing the operations offered by the graphical user interface. The following passages substantiate the reasons and decisions for the successive development of the SPAR-Manager tool.

3.4.1. Implementation Design Decisions

The next step before starting with the actual implementation is to decide how to provide the program to the users. Since the programming language is predetermined, there is the choice in between a variety of JAVA technologies:

- A standalone application with a SWING based graphical user interface.
- A Java Applet for remote use.
- A Java Servlet with JSP based user interface.
- An Eclipse plug-in.

The first and last do not provide directly the potential for remote or web-based access. But the data processing, following the MVC pattern, should not be affected by the concrete way of access anyway. So this part of the program could be, with an appropriate modular design, supplemented with a remote interface at a later point in time. Actually, to close the circle in the programming model, a deployment as web service would be possible.

Furthermore a web-based access should not be separated from other management functions of the corresponding workflow machine. In the opposite functions operating on essential files, should be integrated seamless into the management interface. This lies out of the scope of this study thesis.

In the near future the tool will be primarily used to advance and test the Stuttgarter Workflow Machine. Due to the fact, that the project was developed on integrated development environments based on Eclipse for example the Rational Application Developer from IBM and is still in progress, there is a high probability that attendant developers will use Eclipse based integrated development environments for further work. Therefore it is obvious to integrate the tool into the Eclipse platform.

The best and easiest way to is a realization as Eclipse plug-in. Additional, the possibilities to benefit from the pre-existing functionality of Eclipse is a reason for an Eclipse based approach. Eclipse Ganymede is a sophisticated platform to develop rich clients. So finally the plug-in concept based on Eclipse as a rich client platform allows a web-based distribution of the tool just as well as a standalone application.

Following that reasons, at this stage of the SWoM project, an Eclipse plug-in seems to be the best choice. Thus the next section gives a short introduction to the Eclipse project and the concept of Eclipse plug-ins.

3.5. The Eclipse Project

From the former section some could receive the impression that Eclipse is tightly associated with Java. As a matter of fact Eclipse is very famous in the world of Java developers. But Eclipse is more. At the beginning Eclipse was the successor of Visual Age for Java. Visual Age was an integrated development environment for a variety of platforms and programming languages. Most of the parts where implemented in Smalltalk one of the first object oriented

languages. Smalltalk was developed and implemented by OTI (Object Technology International).

OTI formally independent joined IBM Canada and implemented the first version of Eclipse. Even today the relationship between Eclipse and professional IBM products like Web Sphere could not be overseen. In November 2001 the Eclipse Consortium was founded to expedite Eclipse as an open source project. 2003 the Eclipse Foundation followed, a charitable organization leading the progress of Eclipse.

The opening to the world wide developer community somehow caused the projects developed with and for Eclipse to diverge. This involved inevitable problems with consistency and compatibility. Therefore since 2006 the key projects were bundled into so called simultaneous releases. Named after the Jupiter satellites there were Callisto, Europa and Ganymede. The latter and accordingly named after the largest moon embraced 23 of these key projects 2008.

For the objectives of this study thesis the voting was for Eclipse last but not least because of the variety of resources suitable for the SPAR-Manager application. Therefore the next section provides a closer look to the plug-in concept of Eclipse and the rich client platform.

3.5.1. Eclipse Plug-ins and Rich Client Development

Starting as an integrated development environment the Eclipse platform, at the outset nearly unintentionally, expanded to a full blown “Software Component Architecture” (SCA). Even before, Eclipse supported a lot of features forming a sophisticated development environment. Eclipse provides an adaptive workbench for more and more kinds of programming and developing projects. Furthermore matching vies depending on the project intentions and many wizards providing an easy initial start and proper structure for a project. In addition eclipse has a number of specialized editors alleviating the work on certain kind of files. And of course Eclipse encompasses the nowadays usual functionality like automatic code extension, code generation, debugging and much more.

But the true power of Eclipse lies in the magnitude of tools. To mention only some, there are tools to monitor the memory usage while programs are running, tools supporting modeling or tools to design a graphical user interface. Foremost Eclipse aimed to be a platform to ease the integration if such development tools to the IDE.

As sketched in the introduction in the meantime, the network technologies advanced rapidly. This also changed the requirements of software buyers. Formerly most applications were directly installed as standalone applications on workstations. This did require a massive effort for software installation, maintaining and updating. Centralized solutions seemed to perform better. So the next steps were software repositories for remote install and update. Yet there was a lot of expense necessary to customize the programs for the individual end users.

Remote access in the other direction, for example to file servers or other services, was done by hand (means typing the appropriate commandos in shell) via special protocols like ftp or telnet. This constrained the net-based interaction to a small group of insiders. On the other hand the average end-user was already accustomed to sophisticated graphical user interfaces disburden them from the operations running behind it.

Nevertheless eminently in the business world was a demand to provide remote access to the end users to databases and other services. As an effect distinguishable network based architectural styles originated. For comprehensive information about this topic the dissertation of Roy Thomas Fielding is recommended. Roy Fielding for example describes the World

Wide Web to be intended as "… Internet-scale distributed hypermedia system, which means considerably more than just geographical dispersion" [FR2000].

One of the reasons for the astonishing success of the World Wide Web is that it relies primarily on simple request response interactions maintaining minimal or no state at all. So called thin clients, the most famous are web browsers, handle first and foremost the correct presentation of hypermedia documents, other data is only temporarily stored not least for security reasons for example as cookies. On the server side only for the time of a session context and state information is preserved. After the interaction completed, in the majority of cases, the participating computers neglect that the interaction has happened at all. Otherwise the whole internet would not be scalable.

This may be sufficient for "Internet-Shopping" or to work on a document remotely. But for more complex net-based applications, for example to work on a product development project from all over the world, the thin clients were not satisfactory. The demand was for customizable applications, distributed and updated over the net, running on the client machine and equipped with an elaborate graphical user interface. This is the point where rich clients enter the game.

Figure 7 shows an overview to the plug-ins building the base for RCP applications. In the leftmost corner OSGi appears. OSGi stands for Open Services Gateway initiative. Behind is a specification for a comprehensive and dynamic component model. Eclipse is a framework implementing this specification. For more information see the home page [OSGI].

The other parts of the picture will be discussed in the next sections.

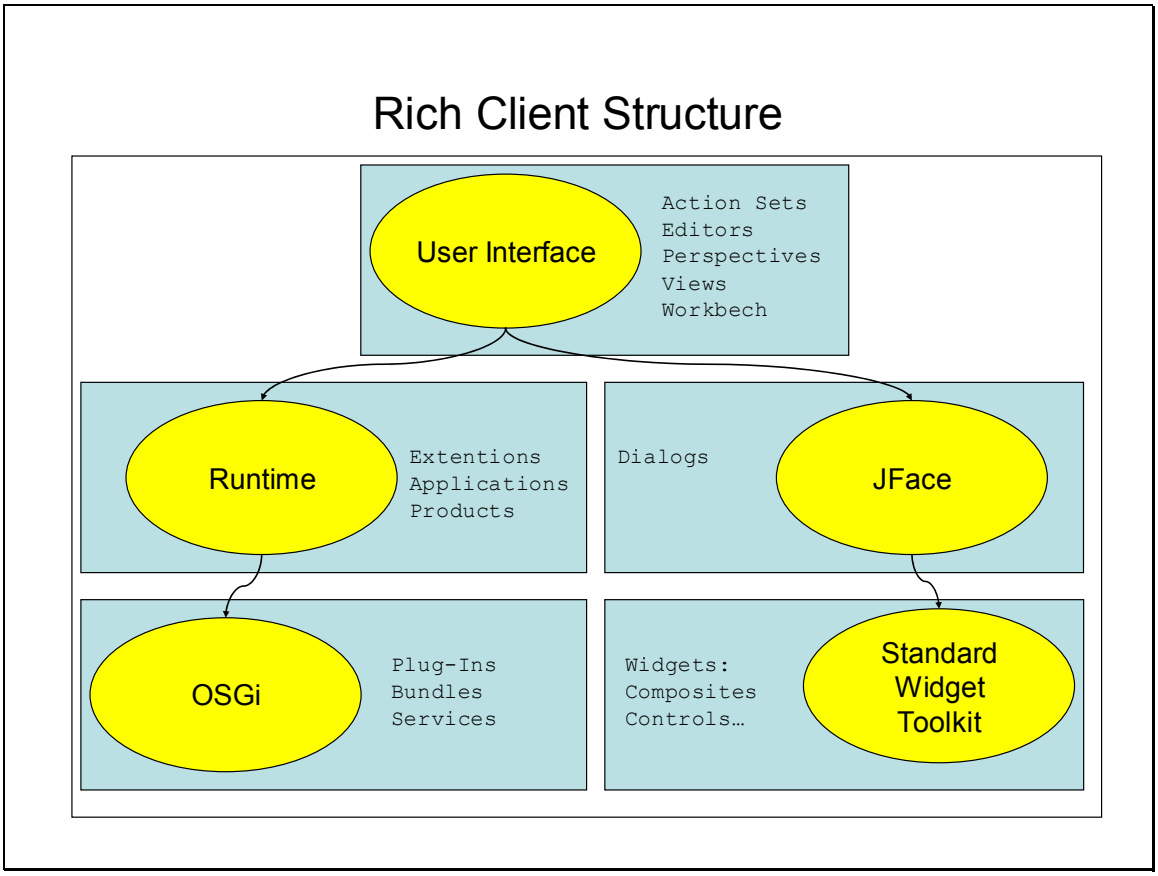

Figure 7: Plug-ins and their Contribution to a Eclipse RCP Application [4]

3.5.2. Standard Widget Toolkit (SWT) and JFace

Rich clients are intended, and therefore the plug-in development environment for rich clients is optimized, to develop sophisticated, interchangeable and reusable end user applications. This is not restrictive. In principle a rich client application can implement everything, for example servers without a user interface at all.

One focus is to support the development of comfortable graphical user interfaces. As mentioned before the developer is not forced to use a special Java graphical user interface framework. Nevertheless the developing tools, wizards and other, of Eclipse plug-ins are tending to JFace on top of SWT. Mainly, but not only because of that, the Spar-Manager GUI will be based on JFace and SWT too. The following gives a brief discussion of the reasons for the decision.

[4]The figure is based on figure 2-2 “1,000-foot RCP view” from [LO2006], page 15.

There are three main Java user interface (UI) toolkits:

- **Abstract Window Toolkit** (AWT), the first and simplest.
- **Swing**, which is based on AWT and offers peerless components
- **JFace** provides an implementation collection for many common UI tasks. JFace uses **SWT**, which provides a set of operating system independent APIs for widgets and graphics.

For further explanations how the toolkits differ please refer to the first chapter of [GJ2005]. The book offers comprehensive information to SWT and JFace.

Even often discussed, aspects like execution speed or quality of the look and feel play a minor role for a comparable small tool like the Spar-Manager. SWT is supposed to be faster, than the other two, but with modern hardware a user will not be able to recognize a difference. Nevertheless it is an advantage of SWT. AWT and SWT both need support by native system code, but SWT reduces the coupling to a minimum. In contrast, Swing does not depend on peer implementations from the underlying windowing system, but SWT supports most of the common operating systems.

The main reason against AWT is the small number of components and: “it’s not suitable for creating full-fledged user interfaces [GJ2005].” Swing compensates this demand. As a matter of fact Swing would be the most platform independent and powerful tool for the design of an individual user interface. But since Spar-Manager is a tool integrated in a complex system it is better to provide the look and feel of interface components the user is used to and expects like SWT does. Furthermore because of the capabilities of Swing a lot of complex programming can be estimated, so SWT and JFace will be a more effective solution.

In a nutshell the main reason for JFace and SWT is that they support a wide variety of sophisticated GUI components, easing the work, and the excellent integration in the Eclipse plug-in development environment used for the implementation of the SPAR-Manager.

4. Implementation

This section describes the concrete implementation of the SPAR-Manager tool. The implementation was performed iteratively, step by step. Therefore the subsections of the chapters are structured following these steps.

4.1. Initial RCP Application

Initial a Ganymede Eclipse SDK 3.4.2 distribution based on a Java Enterprise Edition SDK 5 from Sun was used as IDE.

For an entry point, a RCP-based "Hello-World" application was generated by one of the wizards Eclipse provides. Next, a view was added by contributing a view extension to the Spar-Manager plug-in project. Figure 8 shows a screenshot of the corresponding "Extensions" page Eclipse provides for the plug-in xml file, describing amongst other the extension details of an RCP plug-in. The added view is selected.

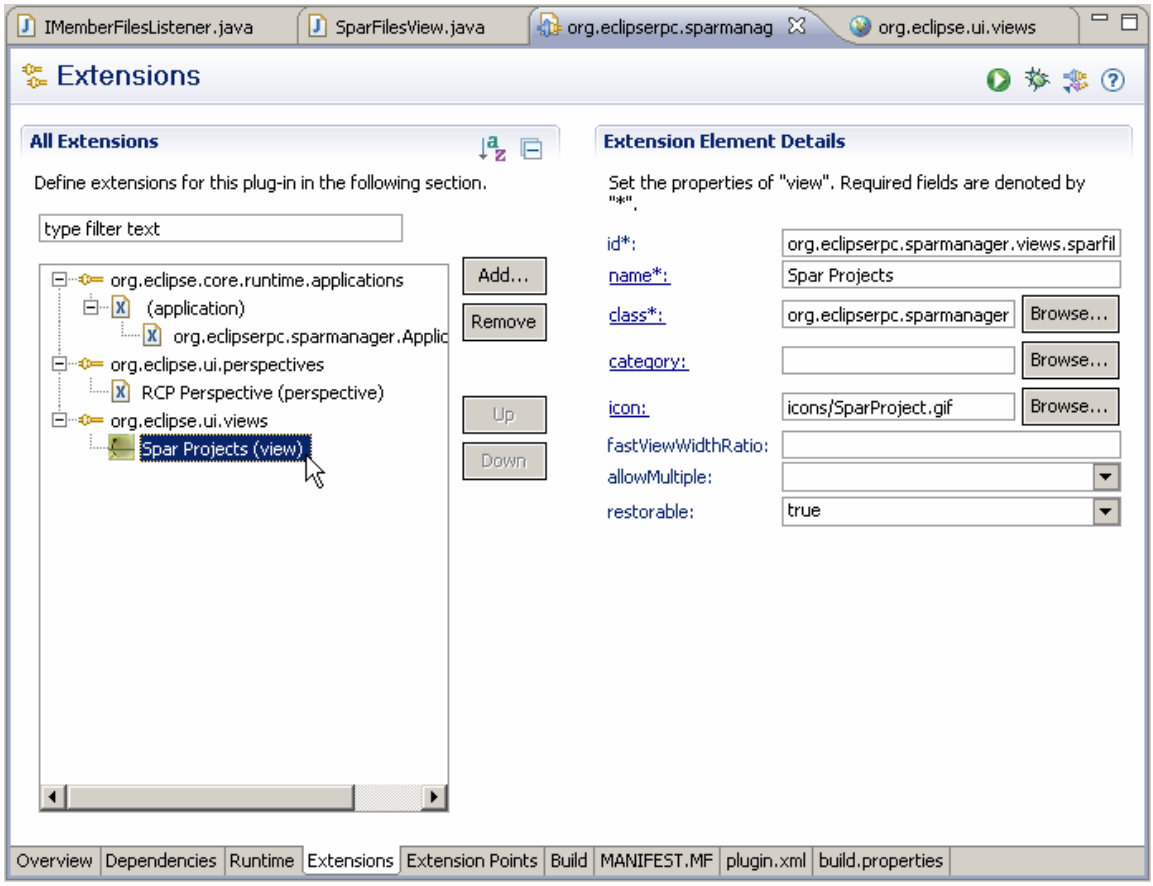

Figure 8: Extensions Page of the initial RCP Plug-In

At this stage (in fact since the generation of the "Hello World" Project) the plug-in is already capable to be executed. The necessary steps were all supported by the Eclipse IDE wizard and resulted in the generation of several Java classes hiding most of the code necessary to provide a graphical user interface. The classes are intended to be supplemented. The next section describes the assignments of the generated java classes.

4.1.1. Generated Classes for the SPAR-Manager RCP Plug-In

- **Appication.java**

The application class is the entry point to a plug-in application. The RPC SDK target the plug-in project uses contains framework libraries, comparable to an installed Java Runtime Environment (JRE). The JRE includes a huge amount of code, but needs a Java program to use the code. What in a Java standalone application is done by implementing a class containing the `main()` method as first entry point is for Eclipse plug-ins the application class.

Since the application class is the main entry and exit point it is a possible position for code that has to be executed before and after any other. The complete listing is not shown, but the class implements at least two methods:

`start(IApplicationContext context)…`

`stop()…`

In earlier Eclipse versions the method `start(..)` was called `run(…)` and no explicit stop method was defined, but the basic functionality remained the same.

In the context of the SPAR-Manager tool the start method creates a display, and then an Eclipse workbench with the display and a new `ApplicationWorkbenchAdvisor` object as parameters. This leads eventually to the opening of a window and simply loops forever handling user generated events (keyboard, mouse…) until the loop is advised to exit or the last window is closed. After that the display is disposed and the workbench closed to free the taken resources.

- **WorkbenchAdvisor.java**

This class was instantiated and passed as the second parameter of the `PlatformUI` method `createAndRunWorkbench(…)` from the start method in the application class. As the name implies the `WorkbenchAdvisor` determines the behavior of the workbench. In the SPAR-Manager implementation the class encapsulates the ID of the initial erspective and provides a method to create a `WorkbenchWindowAdvisor` resulting in a new instance of an `ApplicationWorkbenchWindowAdvisor` the next class in the listing.

As parameter it has a `configurer`. The class contains an overridden method to initialize this `configurer`, which is used here to call the `setSaveAndRestore(…)` method of the `configurer` with true, which results in the behaviour, that changes made on the main SPAR-Manager window, for example resize it, are remembered and the application looks accordingly after the next start.

Additional the `configurer` can be used to instruct the application not to terminate the main event loop when the last window has been closed. This enables applications to run in the background, even after all related windows have been closed. Maybe useful for some scenarios, such behavior is not expected by the user and therefore offends the rules 4, 7 and 8 from Shneiderman (see section 3.2).

The perspective ID provides a human readable identification for an extension (see Figure 8, the second entry in the tree view). The extension identifier specifies a class (the last class in this list) implementing a view for the application. The `Perspective.java` file defines the layout of the initial perspective.

- **ApplicationWorkbenchWindowAdvisor.java**

The `ApplicationWorkbenchWindowAdvisor` class controls the look and feel of a window used by the application. At this stage of the project only the `preWindowOpen()` method was used, again via an `configurer` to determine the initial layout of the main SPAR-Manager window, like size and application name. Other possible methods like `postWindowCreate()` will may be interesting later on. More important is the `createActionBarAdvisor` instantiating and returning an instance of an `ApplicationActionBarAdvisor` the next class in the listing.

- **ApplicationActionBarAdvisor.java**

Beside the methods `fillMenuBar()` and `fillCoolBar()` determining the content of the action bars the class creates the actions related to the bar entries in the `makeActions()` method. Additional the actions are registered. More to actions will be described in section 4.3.2.

- **Perspecitve.java**

As mentioned before this class correlates the initial perspective with a layout. So at this stage the sole method `createInitialLayout()` is used to indicate that later on there will be an editor area (`layout.setEdirotArea(true)`), and to add a view. Again this is the next class in the listing.

- **SparFilesView.java**

The `SparFilesView` class provides the preliminary access to the actual data model of the SPAR-Manager implementation project. Furthermore the presentation of SPAR projects and the contained files as tree viewer is maintained. Since the SPAR-Manager is designed as an Eclipse plug-in the users interacts with the application through views and editors. Perspectives (the class above) are used to arrange views and editors in a customizable way. For now only one view is used to present and work on SPAR projects and their embodied files.

At this stage of the project in the `SparFilesView` the method `createPartControl(…)` is used to instantiate a tree viewer. The tree viewer is then supplied with an entry point to the data model. Details to the tree viewer are provided in section 4.3.1. The next section introduces an initial data model for SPAR-Manager projects and their structure. For testing purposes the data model was endowed with some fake projects by the `initializeProjectTree(…)` method.

4.2. Spar-Manager Architecture

The formerly section described the initial generated skeleton for the application. Step by step the framework was extended by additional components forming the application. This section will give a deeper insight in the architecture of the Spar-Manager application and how the components are acting together.

One principle for the design of the Spar-Manager application was the MVC model pattern. Although a strict data model for Spar-Manager cannot be distinguished. Eventually the Spar-Manager operates on documents stored as physical files somewhere. To provide operating system independence the implementation uses a virtual resource model representing and mapping the system dependent file information to real files.

That virtual model plays not only the role of a data model for the view, but also acts as a mediator and controller, invoking the necessary operations at another abstraction layer dealing with the “real” physical files.

The implementation was designed to separate program parts with different tasks. Figure 9 shows an overview of the coarse grained architecture of the application, which will be refined in more detail for some of the components.

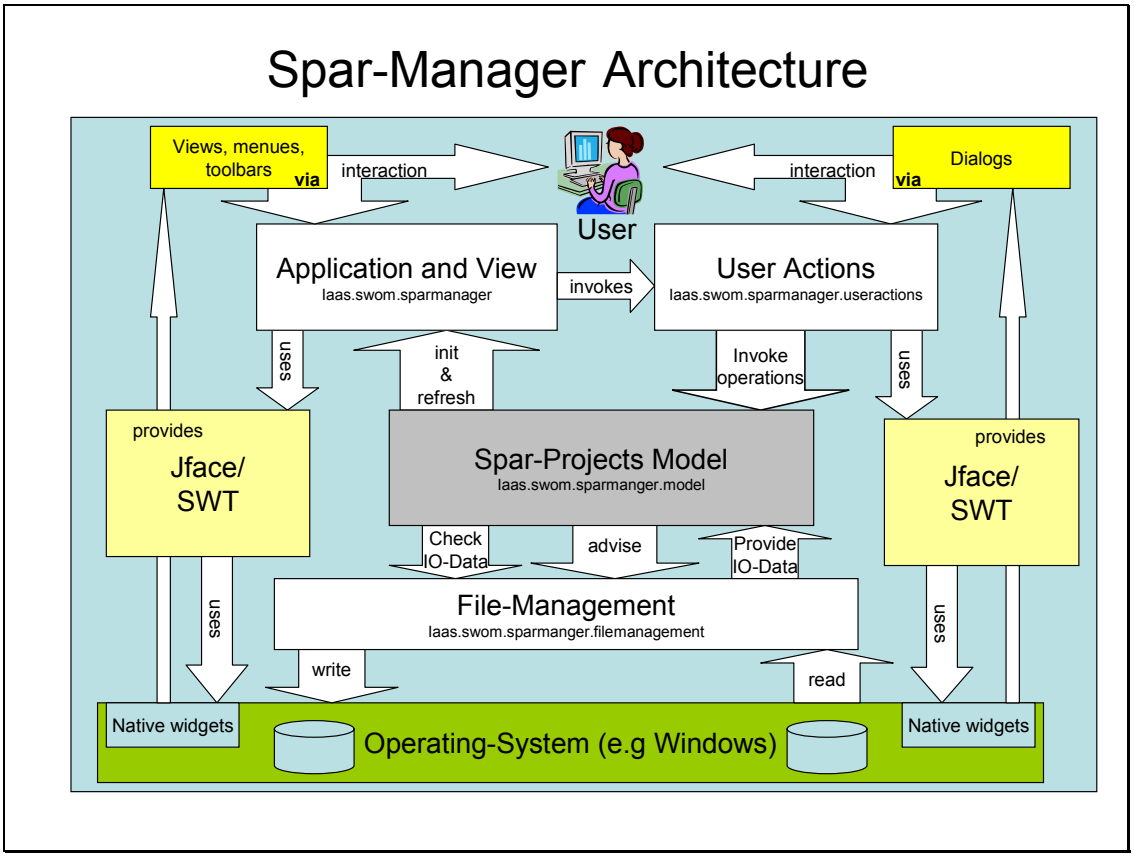

Figure 9: Coarse grained Spar-Manager Architecture

The components are represented by the white and grey boxes. Under the description of a component the java package containing the implementation is mentioned. The component structure shown here is reflected in the actual implementation structure. The colored boxes represent the underlying operating system (green) and the Java GUI libraries used (yellow).

Primary the user interacts with the tool via the main view. The main view also presents a presentation or the SPAR projects via a tree viewer. By clicking tool bar icons or menu entries the user can invoke actions manipulating the underlying SPAR projects model depending on the selected item from the tree viewer. The distinction between the main view and user actions is not necessary and was made only for clarity. Therefore both are settled at the same level in the architecture.

In the middle the components reside, bridging the gap between the underlying operating system and the application. Right and left the GUI libraries used to create views and dialogs. These are sophisticated libraries provided by Swing and JFace. As discussed in section 4.5.2 the libraries provide dialogs and GUI components from the underlying windowing system and therefore present the look and feel the user is used to. For example file- or directory-browsing dialogs from Windows.

In the center the grey box represents the abstract, virtual, model representing the structure and content of the SPAR projects handled by the tool. To be independent from the physical files represented by the model an additional abstraction level was added. The SPAR projects model is not operating directly with the files of the underlying operation system. In fact the model should not be bothered with dependencies to the operating system at all. It is responsible to initiate, manage and refresh the view according to the content of the SPAR projects. Furthermore it handles changes made on projects by user actions and refreshes the model accordingly.

To shield the model from the native file system, it invokes additional classes for file management, which perform the necessary operations on the native file system. The code to handle the zipping of the content into an archive (SPAR file) can also be found there.

Because the SPAR project model has to be synchronized with the physical files additional measurements have to be taken. Particularly some kind of mapping from the virtual content model to the origin files and their copies has to be done.

Therefore the following sections describe the interaction of the main architecture components more detailed.

4.2.1. Prototype Data Model

Hence the Spar-Manager plug-in shall support human users composing the files for a complete process model description the primary intention of the prototype data model is to provide an adaptable and appropriate structure for the files.

The Spar-Manger tool is designed to handle an arbitrary structure. The main entry point for a process model description composition is a SPAR project. Figure 10 shows the general assembly of a SPAR project. The Project contains a set, actual in code an `ArrayList`, of the process model related files, which are not WSDL files. These are files corresponding to the “Programming in the Large” part of the programming model. In figure 10 the BPEL file is in the foreground.

Furthermore the project contains one or more folders, which also can include additional sets of documents. The WSDL files associated with the “Programming in the Small” Part of the programming model are assembled in the WSDL folder. There is at leas one WSDL file associated with the process in that folder too. Although there are process models thinkable, which do not rely on additional WSDL files, the process WSDL file must be available. Therefore the WSDL folder is generated always automatically, even for empty projects.

To keep the model open and prepared for changes or reuse later the description above is not strict. So the prototype model can operate on arbitrary files. Furthermore the user can add additional folders to hold other files that may be related with the SPAR project. It would be also possible to extend the structure by deeper hierarchy levels, allowing for example subfolder.

The prototype model of the SPAR projects can be seen as the central part of the application, where the other components handling user actions or the presentation of a SPAR project come together.

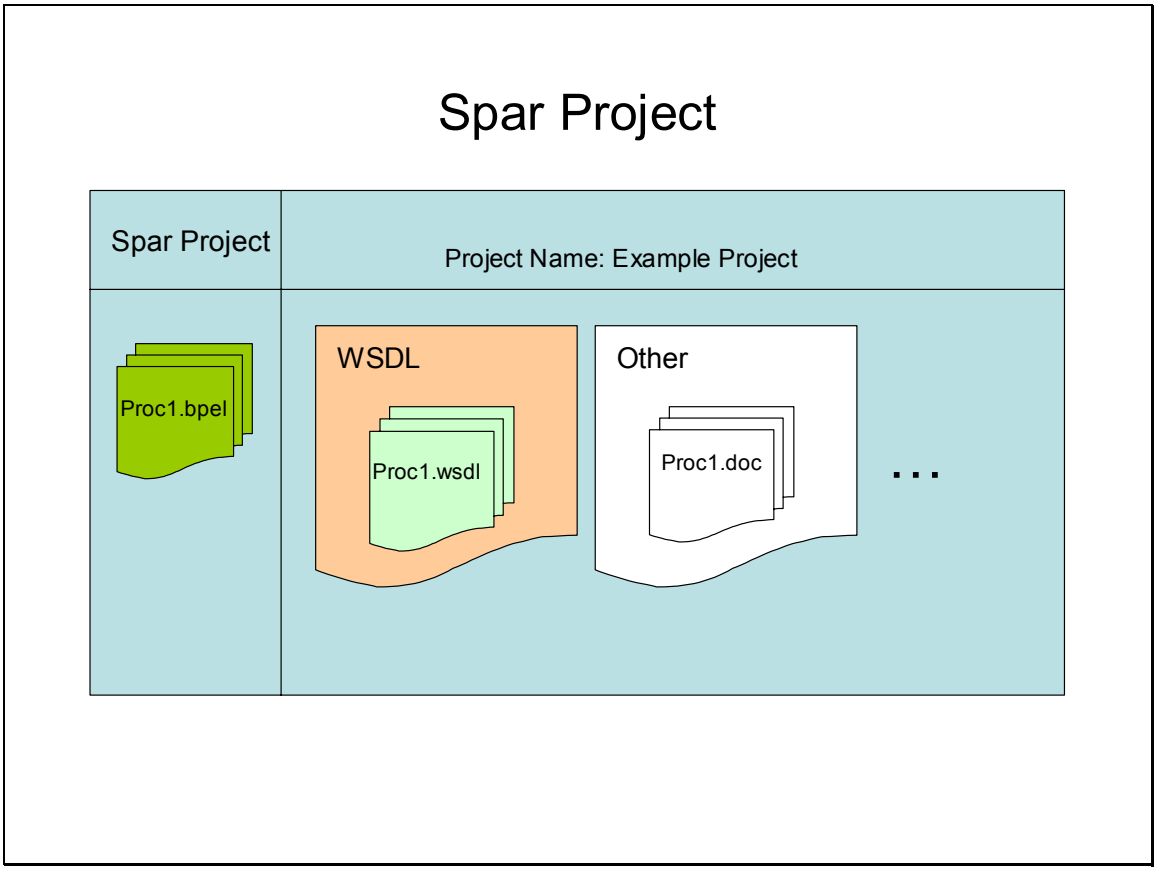

Figure 10: Assembly of a SPAR Project

The implemented classes handling the structure and manipulation of SPAR projects are aggregated in an extra Java package. They provide the data structures and methods to manage operations on the SPAR projects. As later described the classes are intended to operate on a virtual model calling methods in separate classes handling the actual manipulation of physical files and folders on disc. They are called by user actions and provide events to refresh the view.

The actual implementation follows a familiar approach to model the content structure like a tree reflecting the structure of the according physical files and folders. SPAR projects contain a list for files and for folders, which again contain a list of files aggregated in that folder.

Variant to the most approaches found in literature, Spar-Manager uses heterogeneous objects for the model. This means, due to there distinguishable nature different class implementations represent projects, project files, project folders and files in folders. Each child knows its parent and vice versa. Therefore it is possible to navigate through the structure.

For the development the tree was restricted to a height of 4 (the abstract root is not included), which means that no subfolders for folders are available, but “deeper” levels can easily supplemented by allowing folders to contain lists of folders. Structures like this have to be

treated with care because they contain inherently the possibility to construct cycles. When a folder contains somewhere deeper in the structure another folder, which was previously declared as a parent folder of the first folder, for example a recursive search over the content may be caught in an endless loop. Due to this risks and because for instantaneous tasks such structures are not necessary the range of substructures was restricted.

Figure 11 shows an overview to the Spar-Manager general tree structure.

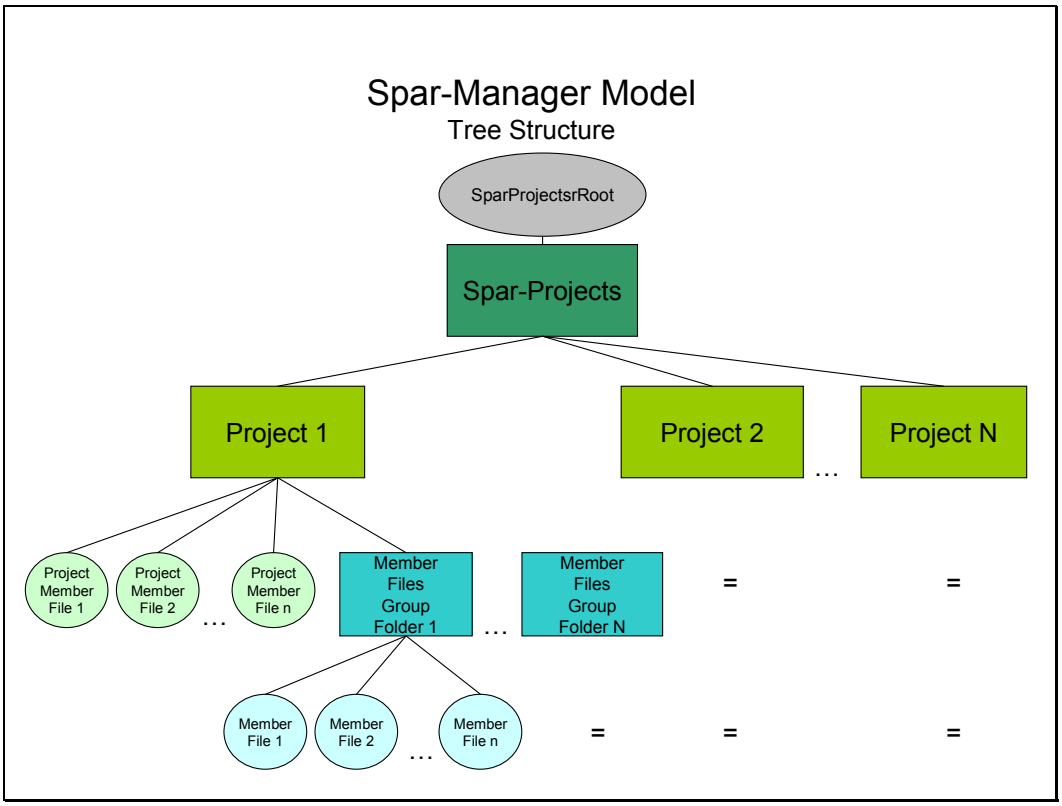

Figure 11: The Spar-Manager Model Structure

The main entry point to the virtual model structure is an abstract class with name `SparManagerRoot`, which is implemented by a singleton, to be reachable all over the application. Physical the structure is reflected by real files on disc. This means each project has an own folder containing the process related files and at least one folder for the WSDL files. To avoid endanger of extern files, which may be used by other applications Spar-Manager operates only on copies, which are created on the fly, while the user composes a SPAR project.

As described later Spar-Manager could be extended by functionality to support the user to ensure that the copies are in sync with their origin files. Figure 12 shows the tree view of a standard SPAR project for testing and a fractioned screenshot of the real content on disc presented by the Windows Explorer.

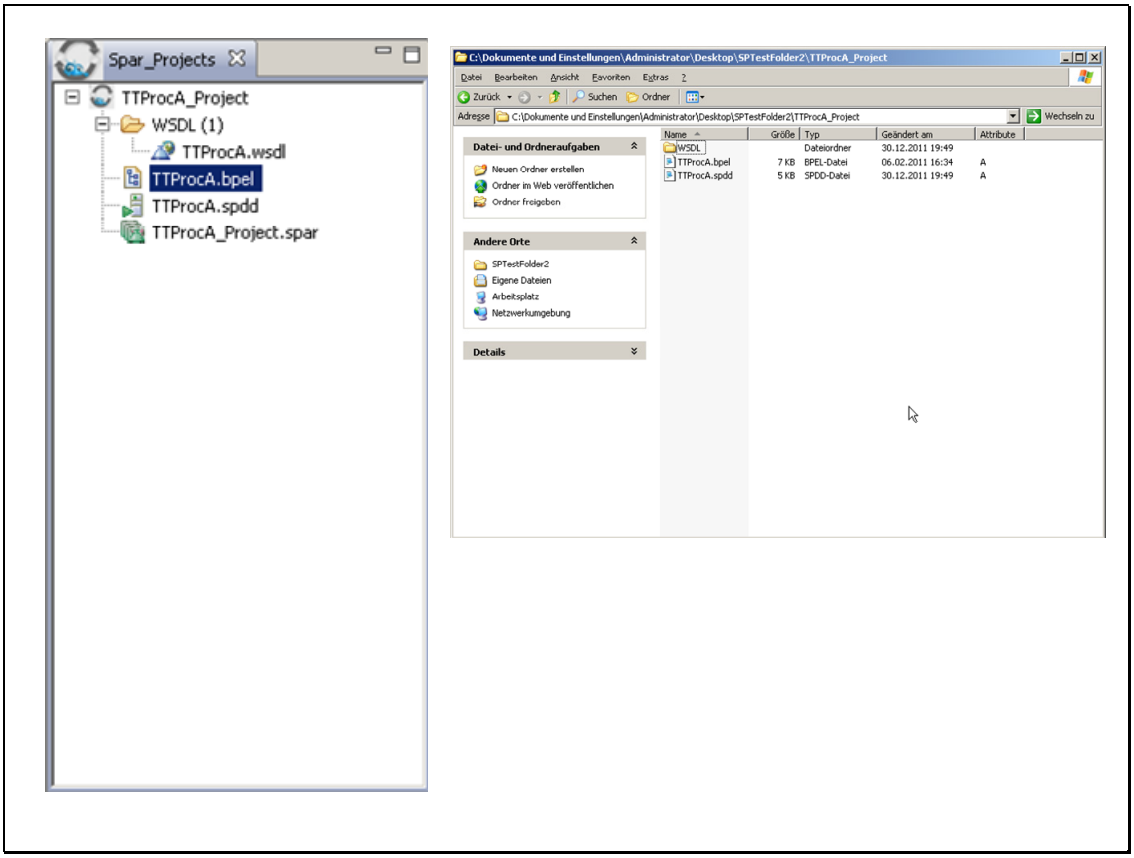

Figure 12: The Tree Viewer and the corresponding Content in the File System

The classes handling the content of the SPAR projects are called by user actions and provide events to refresh the view. The next sections describe the handling of user initiated actions and the presentation

4.3. Basic Functionality and Presentation Implementation

This section gives an overview for the general components Spar-Manager comprises to interact with the user. There are first the main window, showing two major sights, one for the view on the data model, and an editor area to display and edit the content of the involved files. In the editor area the internal editors are displayed.

4.3.1. The main View and Editor Area

The main window of Spar-Manager is composed by a main view and an editor area both of them are determined by the perspective of the Spar-Manager plug-in. The perspective provides a set of layout hints for an application window. For Eclipse plug-ins these are `IWorkbenchWindows`. An `IWorkbenchWindow` has one page. This page can possess editor and view instances.

The Perspective determines where and if editor instances are shown at all. For the Spar-Manager application there is only one view on the left side and an editor area at the right side. Figure 13 shows a screenshot of the application with one editor window open.

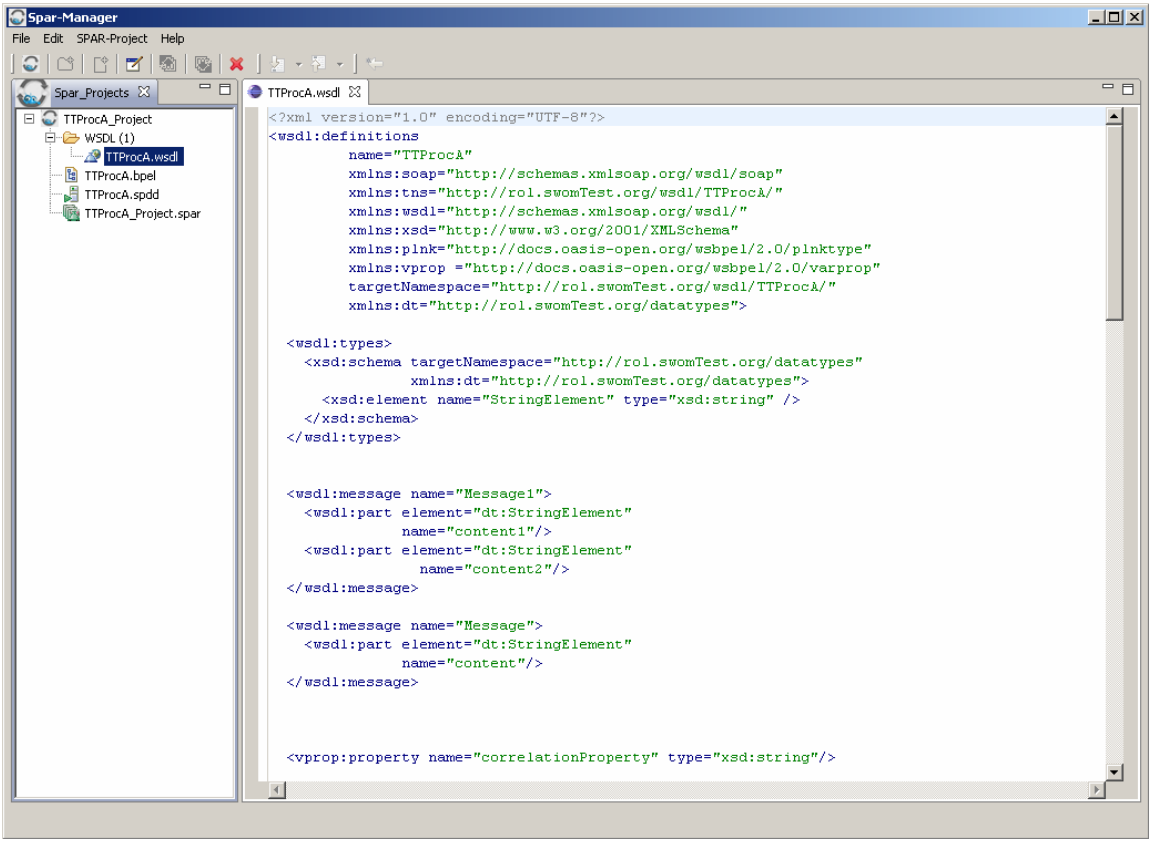

Figure 13: Main Window of the Spar-Manager Application

For the first Spar-Manager version the editor area is not used directly. Files are opened by external editors determined by the operating system depending on the file types for example Windows associates with the type. For later use a specialized XML editor is thinkable for editing and validating the somehow special XML files describing a business process model.

The view contains the tree viewer showing the content of the SPAR projects. The tree viewer is not only to inform the user about content and let him navigate through the SPAR project content via expanding and collapsing SPAR projects or folders.

Furthermore the tree viewer serves as a selection source for the user initiated actions. The code directly concerning the tree viewer is located in the `SparFilesView` class. The initial layout is determined by the advisor classes. For the primary implementation of the Spar-Manager tool the `ApplicationActionBarAdvisor` is of interest. This class contains the entry points for user actions via menus and icons in the `ToolBar` providing shortcuts for experienced users. These actions and how their effect is reflected by the tree viewer will be discussed in the next section.

4.3.2. Actions and Listeners

The actions an user executes, mainly via the mouse and sometimes by typing text, manipulate the content of SPAR projects. The effect of user actions like adding or deleting files or folders or create new ones has to be reflected in the content view of the data model. For Spar-Manager this means the tree viewer has to be refreshed accordingly. For that purpose every class of the virtual file model provides listener interfaces which can be implemented by other classes interested in changes of this model part. Hence Spar-Manager has a heterogeneous

virtual model distinguishing `SparProjects`, `ProjectMemberFiles`, `MemberFilesGroups` (folders) and `MemberFiles` each parent class implements and handles the listener for each child.

If an instance is touched the parent class is informed that a change occurred. At this way changes are propagated up through the content tree and are finally reaching the tree viewer, which implements the `ISparProjectListener Interface`. The implementation of a listener determines what shall be done when the event the listener is observing occurs. For the tree viewer this means just the call of its refresh method.

The tree viewer needs a content provider to know, which and how objects should be presented in the tree. For this reason a tree viewer needs a content provider providing methods to query the structure, like `getChildren(Object parentElement)`. These methods are implemented by the virtual file model.

To avoid pollution of the model with user interface concerns the Spar-Manager application uses an Eclipse adapter mechanism. The workbench provides a so called `BaseWorkbenchContentProvider`, which is able to navigate an adapter type implementing the navigation methods and methods determining how the objects are shown in the tree like `getLabel()`, which will be the name displayed for an model object. Furthermore the adapter provides the associated icons via `ImageDescriptors`.

The adapters for each object encompassed by the virtual file model from Spar-Manager are implemented in a factory returning adapters depending on the type of object. These adapters propagate the navigation methods to the actual virtual file model methods. To enable this flexible mechanism the adapters have to be registered with Eclipse via the platform. For a SPAR project the code in the `SparFilesView` looks like this:
`Platform.getAdapterManager().registerAdapter(adapterFactory.SparProject.class).`

For further information how the user, view, tree viewer and content provider work together see [LO2006].
In the other direction the virtual file model provides methods the action classes can invoke to add, delete and create virtual file model objects. Concurrent the listeners are managed. The action classes are extending the action class. The real work is done by the `run()` method calling the corresponding methods on the selected object of the virtual file model. The necessary user input, for example project names, is provided by classes implementing the SWT and JFace composites handling the display of dialogs and providing the user input via get methods.

The dialog classes are called by correspondent action classes, which are implementing the `ISelectionListenerInterface` to obtain the object from the virtual file model the user has currently selected. The selection determines also, which action are available for the selected object.

The actions themselves are managed in the `ApplicationActionBarAdvisor` class. This class maintains the creation of actions for a window and populates the menu, toolbar and status line of that window. One of the advantages of an advisor is, to enable the developer to create actions “before” they are added to the corresponding window. Therefore the same action can be added at several locations.

The same actions are provided via the window menu bar menus and as shortcuts presented by icons on the toolbar. The menus are handled by menu managers, which are keeping track of

actions and ease the composition of nested menus. For more detailed information about actions and managers and the corresponding windows please refer to [LO2006].

4.3.3. Spar-Manager Options

For different user intentions the coarse grained program behavior should be adjustable. This is normally done via options. For example the user can decide if he wants to choose an editor to open a file or use always the same editor. Such general adjustments should be preserved for the next startup of the program.

The Spar-Manager application serializes the option information in a permanently stored XML file in the installation directory. This XML file is designed to be dynamically extendable for a hierarchy of options. In the current version only so called “DualChoice” options were used. These options allow the user to enable or disable a certain behavior.

The correspondent options dialog is dynamically created according to the content of the XML options file. For each options entry a SWT tab item object is created. The tab items are arranged on a SWT `TabFolder` opened by the options menu entry in the menu bar. The tab items represent general concerns of the program behavior like options for the creation of the ZIP archive. A level deeper for each tab item an array of option instances is arranged in a `RowLayout` divided by separators. To add additional options just the XML file has to be edited. The tap item labels description and buttons will be created based on this information automatically. The “DualChoice” options for each tab item are presented as a SWT check button.

Figure 14 shows a screenshot of the dialog. When the user confirms the selections they are stored in the XML file and the file is saved.

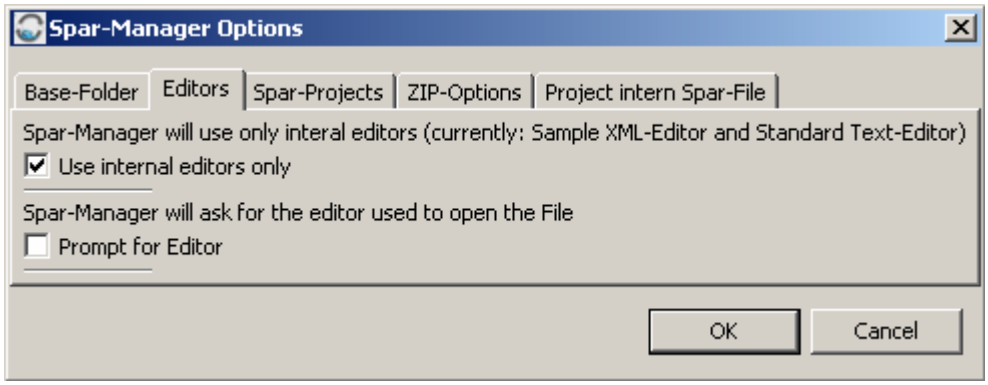

Figure 14: Spar-Manager Options Dialog

To produce and manipulate the options XML file the XMLBeans technology by the ApacheXML project was used. The technology provides ways to access XML document instances by binding it to types. For the Spar-Manager options an XML Schema document was designed in a eclipse editor and compiled by XMLBeans to classes that can be used to access a corresponding XML document. The document was created with help from an Eclipse wizard from the compiled XML schema. The corresponding classes are accessed by a singleton class to read the information needed to create the options dialog and store the user decisions in the same XML file permanently.

The created classes had to be linked explicit to the Spar-Manager plug-in bundle classpath. This was done by the plug-in xml editor of eclipse on the runtime tab-page resulting in

corresponding entries in the build-properties and MANIFEST.MF file of the plug-in. In advanced versions of Spar-Manager XMLBeans may be also used for validation of the files contained in SPAR projects. For more information on XMLBeans see the homepage [XMLB].

4.3.4. Internal and external Editors

Depending on the selection at the options editor page the user can utilize different editors. For the basic version of Spar-Manager there are internal and external editors available. The internal editors were provided by Eclipse.

First a standard text editor for files which are not associated with XML and second a Simple XML editor as plug-in with the corresponding classes created by an Eclipse wizard. Both editors provide the basic functionality of any editor (cut, copy, paste…) the XML editor additional syntax highlighting for XML. The generated classes could be used for advanced and customized functions, supporting the user to edit workflow related files.

Furthermore the user can decide to select an external editor from a list of programs installed at the operating system. Figure 15 shows an example of the dialog.

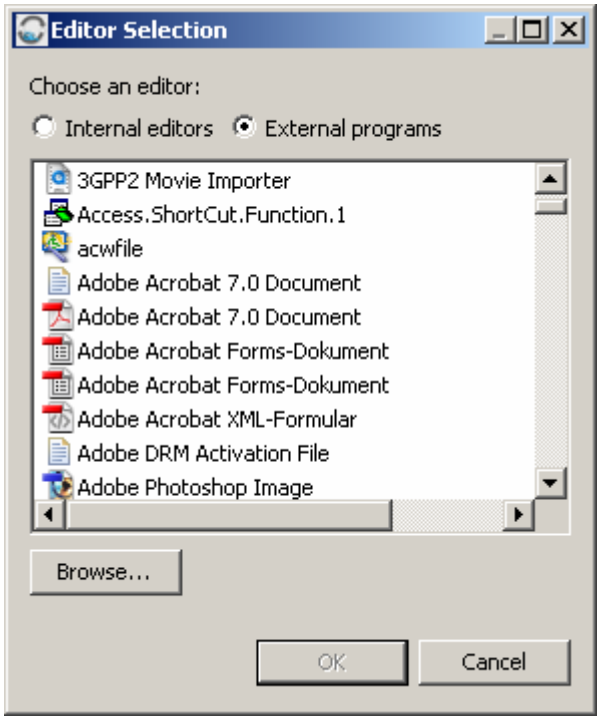

Figure 15: JFace choose Editor Dialog

If the internal and the prompt option is disabled the Spar-Manager will try to open an editor associated by the underlying operating system with the file type.

The editor concept of Eclipse used for the Spar-Manager plug-in requires an input document via the Eclipse file system (EFS). For example the standard XML editor plug-in requires an `IFile` object.

For more information to the eclipse file system two articles [BA2011] and [LO2006] are recommended.

4.3.5. Drag and Drop Support

A very convenient method to deal with files is drag and drop. For the SPAR-Manager application it would be useful to provide the opportunity to arrange the files in a SPAR project by simply dragging them from the desktop or elsewhere for example the Windows Explorer into a SPAR project.

For this reason drop support was added to the Spar-Manager tree viewer using the `addDropSupport(…)` method. That method requires an array of `TransferTypes` for the objects that should be dragged and dropped. The SPAR-Manager uses a predefined `TransferType` named `FileTransfer`, which contains for Windows an array of absolute path names for the files selected by the user. The files can then be dragged by holding the left mouse button pressed into the SPAR project releasing the button.

To init a drop action the SWT `DropTargetListener` interface was overridden, which provides the option to override various events. The array of file paths is embedded in the event data of the `TypedEvent`, which is an input parameter of the `drop(DropTargetEvent dropTargetEvent)` method.

For further information on drag and drop with JFace and SWT the articles [IV2003] and [AJ2003] are recommended.

4.3.6. SPAR-Manager Product and Windows Installer

Eclipse provides a sophisticated and convenient way to export an eclipse rich client application as an executable program called product. With a wizard a file with the type “Product Configuration” can be created. A multi-page editor provides many possibilities for the “Branding” of the application like a “Splash Screen” or an “About Dialog”.

The “Overview” page of the editor provides a link to the Eclipse product export wizard, which is capable to create a ZIP compressed archive or an uncompressed executable composition of necessary files into a directory. The wizard supports a variety of operating systems like Linux or “macosx”. Exporting for example for “win32” the Eclipse subfolder of the created composition includes an *.exe file.

The unpacked ZIP archive, as well as the uncompressed product, was executable on Windows XP, Windows Server 2003 and Windows 7 on a 64 bit laptop without problems.

The Eclipse SPAR-Manager product is executable without further installation issues. But since the Windows user is accustomed to install an executable program on his system a “Setup-“executable for the application was created.

This was done with the support of an Eclipse plug-in offered by NSIS (Nullsoft Scriptable Install System) an open source system to create windows installers. After installation of the plug-in on a Galileo Eclipse 3.5.2 NSIS provides a wizard to create a customizable script. That script can be used to compile a windows executable setup program encompassing the necessary files for a RCP application.

Windows Installers are a complex subject of its own. For inside information see for example [AK2009]. Based on an exported Eclipse product the compiled installers worked without problems on Windows Systems up to Windows 7.

4.4. Electronic Documentation

The CD that accompanies the study thesis includes the Eclipse programming source, executable programs and the document as PDF (Portable Dodument Format) and Microsoft Word file. In particular the following resources are included:

- **abstract.txt**
  A short summary of the study thesis
- **Tobias_Rohm_Studienarbeit_2309_Spar_File_Manager.pdf**
  The document as PDF file.

Furthermore the CD includes the following subfolders:

- **implementation/**
  The folder includes the Eclipse programming project as ZIP archive and the uncompressed workspace used for the development.
- **document/**
  The folder includes the thesis as Microsoft Word file (Tobias_Rohm_Studienarbeit_2309_Spar_File_Manager.doc)

  Furthermore the references as PDF (Tobias_Rohm_Studienarbeit_2309_References.pdf)
- **program/**
  The folder includes the SparmanagerSetup.exe for the most Windows Versions. Furthermore the application as RCP product in the subfolder Spar-Manager 1.0.8/ The Windows executable (sparmanager.exe) can be found in the eclipse/ subfolder.

For detailed install or import instructions please refer to the README.txt files included in the corresponding subfolders.

5. Conclusions

Hence the main assignment of the study thesis was to develop a tool the conclusions are related foremost to practical aspects of software development.

Nevertheless working with what seem to be simple read and write operations on files at first look revealed the importance of consistency and synchronizing considerations. For that purposes this and future applications should profit more intensively from the possibilities of the “Eclipse File System” (EFS) and this right from the start. Surely it is also worth taking a closer look at the editor concept of Eclipse.

XMLBeans have proved to be an easy and straightforward technology for accessing XML files. Unfortunately the timing constraints of the study thesis did not allow for exploring all features.

Working with Eclipse as an integrated development environment proved itself all the time. At the beginning maybe overwhelming Eclipse provides everything needed for professional development.

The plug-in framework is deliberated and powerful. For beginners it might be useful to visit the web page [PG2009] for troubleshooting.

Obviously the design and architecture decisions even for a comparatively little tool like the SPAR-Manager are of high importance. Premature decisions can lead to a lot of additional work at later stages of the project. The rich client skeleton provided by Eclipse is a good starting point.

While developing the SPAR-Manager application the possibility for future extensions were considered in the basic implementation. The next section describes useful possible enhancements for the software.

6. Possible Implementation Enhancements

Improvement is nearly always possible. The SPAR-Manger application is no exemption. Therefore the next section is about some proposals how the application could be meaningful enhanced.

6.1. XML Representation of the File Model

The current version of the SPAR-Manager tool simply reads a folder from the file system and interprets the content as SPAR project. For basic functionality this behavior is sufficient. For more sophisticated usage patterns an extended project management would be eligible. An approach could be to associate a XML file to each project containing the information about the project content.

This meta-information could encompass data about the files the SPAR-Manager expects to be in the corresponding folders on disc. Furthermore it would be possible to keep track of the origin files that where copied to the project and whether they have changed in the meantime or if the copies are still in "sync" with the origin files.

Such an approach would allow advanced usage patterns, for example to prompt the user if the files should be refreshed when they had been changed since the last work on the SPAR project.

Mapping the files to stable resource information could also provide access to distributed files. Eventually the resource files would be independent from their actual location. As a first step the SPAR projects could be independent from a single entry point in the file system (currently the base folder of the SPAR-Manager). For a higher degree of flexibility SPAR project could be composed by distributed resource files, which are only referenced in the project meta-information.

6.2. Submit to the Stuttgarter Workflfow Machine

Hence the product of the SPAR-Manager tool, an archive or SPAR file is designated to be deployed on the Stuttgarter workflow machine a mechanism to ease that operation would be meaningful.

A simple alternative would be to export the file to a distinguished folder, where the workflow machine could fetch the file. More elegant and compliant to the programming model of the SWoM would be using a web service and embed the SPAR file in a message targeted to the workflow machine.

6.3. Validation

Actual the SPAR-Manager has the editor functionality to view and edit the contained XML (and other) files. As a first step the simple XML editor could be customized for the demand of workflow related XML files, for example a special distinctive syntax highlighting for particular elements.

Another valuable feature would be the possibility to verify the syntactical and structural correctness of the files against their corresponding XML Schema files. The XMLBeans technology provides already the necessary libraries supporting this demand.

7. Summary and Outlook

The current state of the SPAR-Manager application can surely be improved and extended. Nevertheless the software can serve as a proper foundation to be enhanced to a more powerful and sophisticated tool.

The basic functionality is covered completely. Irrespective of the underlying file system the architecture is ready for changes and extensions of components. The tool can be used to create arbitrary file compositions and to export them as ZIP archives. While the main focus was to handle workflow associated files in XML.

With the support of the Eclipse integrated development environment the application implemented as a rich client provides a graphical user interface, which is convenient and professional looking.

The application can easily be installed on all Windows versions, regardless if targeted for a 32 or 64 bit system. The Eclipse project can be exported as a "Product" for multiple standard platforms and imported on the actual versions of Galileo or Ganymede Eclipse.

The Eclipse plug-in and rich client platform concept (RCP) combined with the Standard Widget Toolkit (SWT) and JFace more than suitable for the development of applications like the Spar-Manager tool.

The orientation and work with Eclipse was worth the time and recommends for a deeper insight into the possibilities of the Eclipse framework.

With further enhancements the Spar-Manager application could be extended to a sophisticated tool. Particular the file management could be extended utilizing the Eclipse file system (EFS) capabilities. For example the caching possibilities would allow for more efficient editing of the workflow related files.

Furthermore the editor functionality could be adapted to the special needs of business process modeling, eventually with graphical design aid.

The final product could be integrated seamlessly into the developing and modeling process of the Stuttgarter Wokflow Machine.

8. References

[AJ2003] Arthorne, John: *Drag and Drop in the Eclipse UI*
Eclipse Corner Article, IBM OTI Labs 2003
http://www.eclipse.org/articles/Article-Workbench-DND/drag_drop.html
Verified at: 15.02.2011

[AK2009] Kerl, Andreas: *Inside Windows Installer 4.5*
Microsoft Press Deutschland 2009

[AT2009] Allweyer, Thomas: *BPMN 2.0*: Business Process Model and Notation
Norderstedt, Books on demand GmbH 2009

[BA2011] Blewitt, Alex: *Eclipse File System.*
EclipseZone
http://www.eclipsezone.com/articles/efs
Verified at: 09.02.2011

[BD2005] Daum, Berthold: *Rich-Client Entwicklung mit Eclipse 3.1*
Heidelberg: dpunkt.verlag GmbH 2005

[BPEL] OASIS: *Web Services Business Process Execution Language Version 2.0*
http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
Verified at 15.02.2011

[FR2000] Fielding, Roy Thomas: *Architectural Styles and the Design of Network-based Software Architectures*
Doctoral dissertation, University of California, Irvine, 2000.

[GJ2005] Guo, Jack Li: *Java Native Interfaces*: with SWT/JFace
Indianapolis, In.: Wiley Publishing, Inc 2005

[IV2003] Irvine Veronica: *Drag and Drop*: Adding Drag and Drop to an SWT Application
Eclipse Corner Article, IBM OTI Labs 2003
http://www.eclipse.org/articles/Article-SWT-DND/DND-in-SWT.html
Verified at 15.02.2011

[LR2000] Leymann, Frank; Roller, H. Dieter: *Production Workflow*: Concepts and Techniques
Jersey: Prentice Hall 2000

[LO2006] Lorimer, R.J.: *Working with the Eclipse File System*
EclipseZone 2006
http://www.eclipsezone.com/eclipse/forums/t83786.html
Verified at: 09.02.2011

[ML2006] McAffer, Jeff; Lemieux, Jean-Michel: *Eclipse Rich Client Platform*
Pearson Education, Inc. 2006

[OMG] Object Management Group/Business Process Management Initiative
http://www.bpmn.org
Validated at 17.02.2011

[OSGI] OSGi Alliance
http://www.osgi.org
Validated at 17.02.2011

[PG2009] Prakash, G.R.: To*p 10 Mistakes in Eclipse Plugin Development*
Eclipse Zone 2009
http://eclipse.dzone.com/articles/top-10-mistakes-eclipse-plugin
Verified at: 16.02.2011

[SP2010] Shneiderman, Ben; Plaisant, Catherine: *Designing the user interface*: Strategies for effective human-computer interaction
Upper Saddle River, NJ: Pearson Addison-Wesley 2010

[SW2011] Institut für Architektur von Anwendungssystemen: *Stuttgarter Workflow Machine*
http://www.iaas.uni-stuttgart.de/forschung/projects/swom
Verified at: 09.02.2011

[TS2007] Tanenbaum, Andrew S.; Steen, Maarten / van: *Distributed Systems*: Principles and paradigms
Upper Saddle River, NJ: Pearson Prentice Hall, 2007

[WSA] W3C: *Web Services Addressing (WS-Addressing)*
http://www.w3.org/Submission/ws-addressing
Verified at: 15.02.2011

[XMLB] The Apache XML Project: *Welcome to XML-Beans*
http://xmlbeans.apache.org
Verified at: 09.02.2011

Erklärung

Hiermit versichere ich, diese Arbeit selbständig
und nur die angegeben Quellen benutzt zu haben.

Stuttgart, 01.03.2011 _____

(Tobias Rohm)