TOWARDS SIMULATION WORKFLOWS WITH BPEL: DERIVING MISSING FEATURES FROM GRICOL

Mirko Sonntag, Katharina Görlach, Dimka Karastoyanova
Institute of Architecture of Application Systems, University of Stuttgart
Universitaetsstrasse 38, 70569 Stuttgart, Germany
{sonntag, goerlach, karastoyanova}@iaas.uni-stuttgart.de

Natalia Currle-Linde
High Performance Computing Center (HLRS), University of Stuttgart
Nobelstrasse 19, 70550 Stuttgart, Germany
linde@hlrs.de

ABSTRACT
In this paper, we investigate the suitability of the general-purpose workflow language BPEL to create executable simulation workflows. We therefore compare BPEL to GriCoL, a graphical language with proven applicability for simulation workflows in Grid environments. We discover a number of incomparable concepts in the two languages. On the one hand, BPEL's unique features in comparison to GriCoL reveal the rationale behind the approach of using BPEL as basis for a simulation workflow language. On the other hand, based on the features of GriCoL, we are able to discuss how to extend BPEL in order to increase its expressiveness for simulation workflows.

KEY WORDS
Simulation tools and languages, Workflow management, BPEL, GriCoL

1. Introduction

Scientists conduct experiments, computations, and simulations with the help of computers. The term *e-Science* was coined for such kinds of applications. A novel research area and an important step in e-Science is the introduction of workflows to support non-computer scientists in creating and executing scientific applications [1]. The application of the workflow technology in e-Science promises to relieve scientists from the overhead of programming the scientific applications, which may be complex combinations of computations, and to automate work. The goal is to facilitate scientists in concentrating on their core research area without wasting time in struggling with IT-related matters.

The term *scientific workflow* encompasses a broad set of applications. A scientific workflow could, for instance, deal with processing images, automating data transformation operations, or analyzing gene sequences. Scientific workflows that are used to perform the simulation of phenomena based on real world models are called *simulation workflows*. Workflows for simulations have specific requirements, e.g. on the coordination of several workflow instances in a shared simulation context.

In the Stuttgart research center for simulation technology (SimTech[1]) there are two groups working in the area of simulation workflows. The High Performance Computer Center (HLRS) developed Grid Concurrent Language (GriCoL) [2], a graphical language to simplify the creation and execution of scientific simulations in a Grid environment. The Institute of Architecture of Application Systems (IAAS) examines traditional workflow concepts (as described in [3] and by the WfMC[2]) and the Business Process Execution Language (BPEL) [4] towards their applicability to design and execute simulation workflows.

The application of GriCoL and its supporting infrastructure SEGL (Science Experimental Grid Laboratory) for scientific experiments was already investigated [5] and proven with the implementation of a simulation for molecular dynamics of proteins [2]. GriCoL is tailored to the scientists' needs and contains concepts that address specific requirements of scientists on a workflow language for simulations.

In this paper, we examine the expressivity of BPEL in order to support simulation workflows. We thereby want to exploit the requirements collected during the development of GriCoL and the concepts that accommodate these requirements. We compare these two languages to identify where BPEL lacks functionality. We discuss which GriCoL concepts are reasonable to be adopted by BPEL when being used for simulation workflows and suggest solutions to close these gaps. Since we want to pursue an engineering solution, we advocate reusing as much as possible of existing BPEL extensions and available infrastructures.

The contributions of the paper can be summarized as follows:

[1] `http://www.simtech.uni-stuttgart.de`
[2] Workflow Management Coalition, `http://www.wfmc.org`

- Bilateral comparison of GriCoL and BPEL
- Argumentation for the usage of BPEL to design and execute scientific workflows in general
- Identification of gaps in BPEL for simulation workflows
- Discussion of these gaps and suggestions about how to close them
- Identification of characteristics special to simulation workflows as compared to business workflows

Our work is driven by the goal of developing a scientific workflow system to create and execute simulations based on existing workflow concepts and technologies. The system will be used in SimTech to model and run multi-scale and multi-physics simulations such as the flow process in porous media or crack initiation and propagation in metals.
In previous work, BPEL's applicability in the field of scientific workflows was already investigated. Barga and Gannon [23] state that BPEL's capabilities of fault tolerance, transaction support, application integration, and collaboration support are advantageous for scientific workflows. The need for tooling extensions is adverted, e.g. to improve the end-user robustness of BPEL modeling tools because they are used by scientists that have no experience with these technologies. Wassermann et al. [24] notice that BPEL is expressive enough to meet the requirements of scientific workflows. But they analyze that BPEL modeling tools lack abstractions tailored to scientists. Therefore, a BPEL modeling tool for scientists, Sedna, is proposed that implements a number of useful modeling features, e.g. workflow macros, plug-ins, or indexed flows. Akram et al. [25] denote BPEL as "best candidate for orchestrating scientific and Grid services". This conclusion is reached after investigating how BPEL meets concrete scientific requirements on workflows. Some missing features of BPEL for the utilization for scientific workflows are identified, e.g. the re-execution of activities (i.e., the modeling of cycles) or triggering event handlers from within BPEL workflows. Similar to our work, these approaches advocate the usage of BPEL for modeling and executing scientific workflows. However, the argumentations are based on scientific workflows in general. We investigate BPEL in the context of simulation workflows which have very particular requirements on a workflow language. These requirements cannot be met by BPEL modeling tool extensions only as proposed in the mentioned approaches. Crucial yet unaddressed concepts on the language level have to be developed and need support by the workflow modeling and runtime environment. In this work, we identify a subset of these concepts.
The remainder of the paper is structured as follows. Section 2 gives background information about BPEL and GriCoL. Section 3 draws a conceptual comparison between these two languages, identifies gaps, and argues why it is reasonable to use BPEL as a starting point for a simulation workflow language. Section 4 addresses limitations of BPEL in comparison to GriCoL and discusses how these limitations can be resolved. Section 5 presents work related towards BPEL's applicability for scientific workflows. Section 6 concludes the paper and identifies issues that are open for future work.

2. Background

In this section, we introduce the languages BPEL and GriCoL and provide the background information needed for the discussions in the subsequent sections.

2.1 BPEL

BPEL [4] is the de facto standard for the definition of machine-executable business processes (aka workflows) and is widely adopted in industry and research. It focuses on the specification of the syntax and operational semantics of the language constructs. The graphical representation of these constructs is out of the scope of the specification and therefore workflow modeling tools that implement the BPEL specification exhibit deviating representations and notations. Since BPEL relies on the Web services (WSs) technology, it is independent of the concrete implementation of invoked services. BPEL provides a recursive aggregation model, i.e. BPEL workflows orchestrate WSs and are again exposed as WS. This fosters modularization and reuse of workflows and the corresponding artefacts.
BPEL supports two approaches for modeling control flow (and mixtures thereof): graph-based modeling is achieved with control flow links connecting activities; block-based modeling is enabled by structured activities that serve as containers for other activities and that implement certain control flow behaviour (e.g. executing child activities in a sequence or repeatedly). Data handling is implemented by `assign` activities that are able to copy variables, literal values, and endpoint references. Data flow is not explicitly modeled by BPEL constructs but variables as data containers and `assign` activities are implicit means to specify data dependencies and manipulation.
Interaction activities (e.g. `invoke`, `receive`) are used to send and receive messages to partners (i.e., WSs). At design time, only the interface of a WS is specified in BPEL enabling the support of several binding strategies (e.g. static or dynamic binding) [6]. WSs can be invoked both synchronously (i.e. blocking) and asynchronously (i.e. non-blocking).
With the help of a fault handling mechanism reaction on faults can be modeled at design time and executed at run time. The concept of compensation helps enabling transactional properties in BPEL, which is particularly useful for transactions spanning multiple activities in long-running scenarios. A key feature of BPEL is its extensibility that allows creating new activity types or customized data handling operations.
The process models can be executed multiple times in terms of workflow instances, which allows for parallel

execution of work.

2.2 Infrastructure for BPEL workflows

There are several competing tools that implement the BPEL specification [7, 8, 9, 10]. All implementations comply with the following conceptual infrastructure showing the components of a BPEL workflow environment (Figure 1).

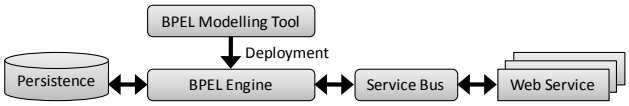

Figure 1: Conceptual infrastructure supporting BPEL

The user typically first gets in touch with the *BPEL modeling tool*. It is used to specify process models either programmatically by editing the BPEL code or graphically using a graphical modeling tool. After modeling, a process model and its WS interface descriptions are deployed on a *BPEL workflow engine*. The deployment requires information that is specified in a format imposed by the engine in terms of a deployment descriptor (e.g. details about the installation of the process model as WS). At run time, the engine creates process model instances, navigates through the workflow graph for each instance, and evaluates expressions and conditions. Usually, the progress of the instances is held persistently to enable a recovery after a server or system failure without the necessity to redo all the tasks already performed by a workflow instance. A *service bus* is employed as the middleware for discovery, selection, binding and invocation of services on behalf of the process instances, and exposes the processes as WSs. Therefore it belongs to the overall supporting infrastructure.

2.3 GriCoL

GriCoL [2] is a graphical language for modeling scientific simulations. Special attention is paid for allowing the user to concentrate primarily on modeling the logic of the experiment without having the executing infrastructure in mind. GriCoL is based on components with predefined internal structure that interact with each other through a defined set of interfaces. The language is of an entirely parallel nature and provides parallel processing of many data sets at all levels: inside simple language components; at the level of more complex language structures; and for the entire experiment. The possibility of parallel execution of operations in all nodes of the experiment program is only limited by the logic of the simulation (e.g. data dependencies that prevent from parallel execution). GriCoL's extensibility allows adding new functional components to the language constructs library.
The language is based on the principle of wrapping functionality in components in order to utilize the capacities of supercomputer applications and to enable interaction with other language elements and structures. That way, program codes written in any language can be wrapped in the standard language component if appropriate adapters for the programs are created.
A main property of GriCoL is the multi-layer principle, i.e. the sub-division of simulations into control flow and data flow. The top level layer (i.e., the control flow layer) is intended for the description of the logical stages of the experiment (see Figure 2a). The main elements of this level are solver and control blocks. Solver blocks represent programs that use numerical methods to solve a part of the overall problem, e.g. a linear equation. Control blocks steer the execution of the simulation, e.g. by evaluating results, choosing paths, or repeating blocks. The sequence of blocks can be specified by two types of lines with different semantics. Serial connection lines (solid in Figure 2a) pass control to the next block only after all runs in the previous block have been finished. Pipeline connections (dashed) transfer control to the next block each time the computation of an individual data set has been completed. That way, control can be transferred many times if many data sets are processed.

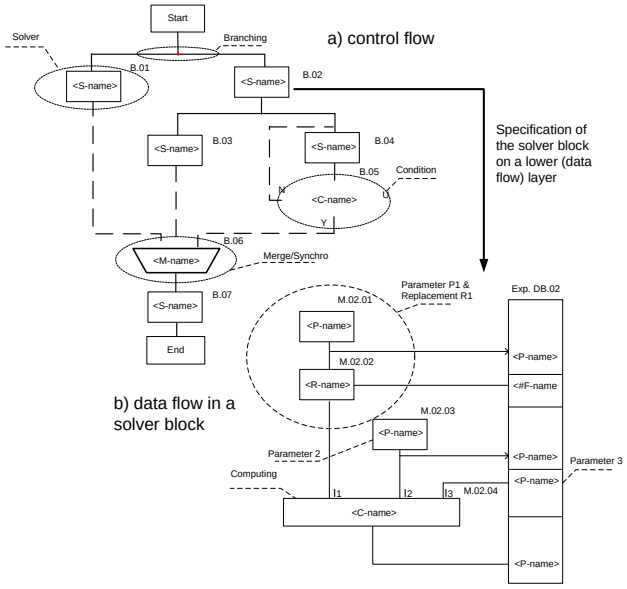

Figure 2: GriCoL – Main Concepts

The bottom level layer (i.e., the data flow layer) is used to specify the behaviour of single blocks and the data dependencies between blocks. Figure 2b shows the specification of a solver block which cyclically computes a large number of input data sets. The main elements of the data flow level are solver and condition modules. Solver modules select data, initialize parameters, start a computation, and update data afterwards. Condition modules select and filter data, evaluate expressions, decide on the continuing execution path, and update data.
Access to data is wrapped by a data repository that abstracts from concrete data sources. That way, data can be used within a workflow independent of its concrete representation on the participating machines (e.g. stored in databases, files).

2.4 SEGL

SEGL [5] is a system for the automated execution of Grid experiments and for the efficient use of Grid resources. SEGL consists of two main components: the application designer is used for designing and debugging of simulation experiments; the application engine controls the execution of simulations on Grid resources. Complex experiment scenarios are realized by the user in the application designer with GriCoL. The technical mapping from this user perspective to the underlying infrastructure is done according to the control flow, the data flow and the repository layers. After the design of the experiment is completed, it is compiled into an executable code and transferred to the application server. The runtime system of SEGL chooses the necessary computer resources. The engine organizes and controls the execution sequence of blocks according to the workflow logic and the contained condition. SEGL steers the experiment while the experiment monitor continuously informs the user about the current status.

3. Conceptual comparison of BPEL and GriCoL

As mentioned earlier, the design of GriCoL was influenced by concrete requirements of scientists. Its applicability for modeling simulations was proven in a field test. Mapping BPEL mechanisms on GriCoL features will reveal concepts important for scientists but currently unaddressed in BPEL.

3.1 Unique features of GriCoL

At first, we have a look at general capabilities of GriCoL and BPEL. Table 1 summarizes our findings. The two languages are designed for different purposes. While BPEL's intention is to express all kinds of business processes in an executable form, GriCoL was devised to create scientific experiments in Grid environments with special attention on dynamical parameter sweeps. GriCoL is strongly connected to its graphical representation. In contrast to this, BPEL does not specifiy a visualization of its constructs. With its selection, update, and filtering functions GriCoL enables handling data directly from within workflows. BPEL externalizes data handling mechanisms to WSs that provide appropriate operations.
As described in Section 2.3, GriCoL workflows are described on two layers, namely control and data flow layers, and contain a data repository that abstracts from concrete data locations. In BPEL, data flow is only implicitly modeled via variables. These variables are filled by messages or assign operations. Thus, they are not connected to data items in a repository. The control flow layer is very similar to process models specified in BPEL.
The control blocks of GriCoL have very similar counterparts in BPEL. Nevertheless, due to GriCoL's pipeline processing the `Merge/Synchro` block can realize complex join behaviour beyond the scope of BPEL (e.g. combining the data of three pipelines in a 1-1-1, 1-1-2, etc. manner and passing it along another pipeline).
The solver blocks of GriCoL are like placeholders the concrete behaviour of which is specified on the data flow layer. BPEL's `scope` can be seen as similar concept that acts as container for other activities and that stands for an atomic unit of work. GriCoL's user solver blocks act as extension point for additional solver block types and hence can be compared to BPEL's extension activity.

Table 1: Comparison of GriCoL and standard BPEL. Shaded cells denote concepts that are beyond the scope of BPEL.

	GriCoL	BPEL
General	Purpose: model simulation workflows independent of the underlying Grid infrastructure	Purpose: creation of machine-executable business processes
	Graphical language	Specification of syntax and semantics, no visualization
	Experiment data handling on language level	Experiment data handling is encapsulated by WSs
Layers	Control flow layer	Process model
	Data flow layer	Implicit data flow through variables
	Data repository	–
Control Blocks	Start	Receive, pick
	End	Reply, invoke
	Branching	Outgoing links
	Merge/Synchro	Incoming links (& empty)
	Condition	see Section 4.3
	Message	Reply, invoke
	User extension	Extension activity
Solver Blocks	Basic	Scope
	User solver block	Extension activity
Links	Pipeline connection	ForEach (if data items in the branches are independent)
	Serial connection (batch)	Flow & links, sequence

As mentioned in Section 2.3 GriCoL allows serial and pipeline connections between blocks. Serial connections can be mapped to BPEL's `sequence` activity or to a `flow` where activities are connected by links. In general, GriCoL's pipelining is beyond the capabilities of BPEL and is therefore further discussed in Section 4.2.
In summary, GriCoL contains four major concepts that are unaddressed in BPEL: the data handling mechanism on a workflow level; the pipeline processing capabilities; the different layers of abstraction/explicit data flow; and the data repository.

3.2 Why using BPEL instead of GriCoL?

GriCoL is specifically designed for scientists to model and execute experiments independent of a programming language and the underlying Grid infrastructure. It provides concepts to maximize the parallelization of the execution of scientific computations. From this point of view it would be reasonable to follow the GriCoL approach.
Despite these facts we advocate the usage of BPEL for

scientific workflows, and enhance it with missing concepts as compared to GriCoL or other languages. This is due to the rich set of advantages BPEL and its supporting infrastructure entail. First, BPEL is an accepted standard in research and industry for expressing executable service compositions. Thus, there already exists an extensive and mature tool support that can be used as basis for further enhancements. Second, BPEL provides concepts not addressed by GriCoL and many other languages for scientific workflows (see Table 2) as we will see in the following.

Table 2: Mapping of BPEL concepts on GriCoL constructs. Shaded cells denote concepts that are beyond the scope of GriCoL.

	GriCoL	BPEL
BPEL Concepts	–	Asynchronous messaging
	Late binding of resources (but list of resources is fix at simulation start)	Late binding of services
	Computation module (non-standard adapters needed)	Application integration through WSs
	–	Recursive aggregation model
	–	Sub-processes (BPEL-SPE)
	Undefined exit of decision module	Fault handling and compensation
	–	Event handling
	No difference between workflow model and instance	Distinction between workflow models and instances
	–	Human involvement / BPEL4People

BPEL and its software infrastructure exploit the full capabilities of the Web services technology, such as an asynchronous communication between partners. Late binding of services allows BPEL workflows to react on a changing software and hardware environment. SEGL also enables late binding of computing resources, but the list of resources that can be utilized for computation has to be provided by the user when starting the simulation [11]. The usage of Web services allows an integration of different tools independent of the programming languages they are written in or the platform they are running on, including Grid resources. Scientists can offer their services to each other and hence enable an interdisciplinary research. With GriCoL different applications can also be integrated into a workflow. Specific non-standard adapters are needed for each type of application, thus impeding collaboration between scientists.

Workflows in BPEL make use of Web services and at the same time are provided as Web service. This recursive programming model enables reusing existing workflows and the collaboration between different scientific institutes seamlessly. The same holds for sub-processes in BPEL [12] with the additional advantage of integrating invoked sub-processes into the parent process's life cycle. Similar concepts cannot be found in GriCoL.

Another advantage is BPEL's powerful fault handling mechanism. It allows catching and handling faults on different language levels (invoke, scope, process level) in a straightforward, user-defined manner. A special fault handling concept is undoing, i.e. compensating, already finished work which introduces the notion of transactions (i.e. units of work) even in long-running workflows. GriCoL also offers an error handling mechanism, but only as an undefined exit of the decision module.

A characteristic of scientific computations is the application of sensors to gather real life data (e.g. air humidity for a weather forecast). Integrating sensor data into BPEL processes is facilitated with an event handling mechanism. This way, sensors that supply pieces of data (as opposed to data stream sensors) can be used in scientific workflows on basis of BPEL. GriCoL is not able to react on external events.

In contrast to GriCoL (and many other scientific workflow languages), BPEL distinguishes between workflow models and workflow instances. This feature eases modeling and execution of simulations: the behaviour of a simulation object (e.g. a molecule) can be modeled once, but is instantiated several times to represent a huge number of these objects. These instances can then be executed in parallel without parallel programming approaches. The results of the instances can be collected and combined by a workflow that represents the simulation context.

There are situations where simulations cannot be executed fully-automated. An active steering on behalf of a scientist is needed (e.g. to decide a refinement of a mesh in a simulation based on the finite element method). BPEL event handlers or the BPEL extension for the integration of human tasks into workflows (BPEL4People) [13] enable dealing with such scenarios natively. In GriCoL, an active participation of humans is not foreseen.

The advantages of BPEL and its supporting infrastructure are numerous. Thus, it is natural to design a simulation workflow language upon BPEL.

4. Closing the identified gaps

So far, we have revealed concepts that are unique to GriCoL when being compared to BPEL. Furthermore, we argued for a simulation workflow language based on BPEL instead of pursuing the GriCoL approach. We now need to discuss, which of GriCoL's unique features should also be addressed by the foreseen simulation workflow language and how to close identified gaps.

4.1 Data handling

On the data flow layer GriCoL offers solver and condition modules that directly operate on external data sources. There are also other approaches that provide access to external data from within the workflow definition, e.g. Kepler with its MoML [14]. Obviously, scientists want to have the possibility for fine-grained data access on the workflow level to select, insert, update, delete and filter data, and to iterate over items of a data set.

Natively, BPEL provides access to external data only via

the communication with WSs. That means a WS needs to be created that is able to access a database or any other data source with appropriate operations (e.g. select, update) in order to load external data into a BPEL workflow. These data sets could then be further processed with the help of `assign` activities that are capable of performing even specialized data transformations. Since BPEL's assignment mechanism with its standard query language XPath [15] is tailored to simple copy operations, it is complex to implement the mentioned data operations.

Table 3: Common data handling operations as implemented by GriCoL and BPEL. Because the BPEL solutions do not directly work on external data they are not semantically equivalent to the GriCoL concepts. Shaded cells denote concepts that are more expressive than in the opposed language.

Data operation	GriCoL	BPEL
Iterate	Parameterization module, replacement module	ForEach & assign, invoke & forEach & assign
Insert	Link to data repository	Assign & invoke
Select	Selecting module	Invoke & assign
Update	Updating module	Assign & invoke
Filter	Filtering module	Assign
Delete	–	Assign & Invoke

Table 3 summarizes how these operations can be realized in GriCoL and BPEL. Besides the deletion of data, GriCoL supports all of the operations with native modules or concepts. Iteration can even be accomplished on internal data by a parameterization module or on external data with the help of a replacement module. In BPEL, the operations are not supported natively. Workarounds with combinations of `forEach`, `invoke`, and `assign` activities are needed. Moreover, since `assign` activities are tailored to copy operations, XPath expressions and XSLT [16] scripts have to be employed to achieve similar results to the GriCoL solution. It needs to be emphasized that the BPEL workarounds are not semantically equivalent to GriCoL's native implementation because the concrete data operations are wrapped by Web service interfaces and do not directly work on external data.

In order to support scientists in setting up their experiments a BPEL modeling tool extension could provide workflow fragments [17] that represent these (and other) data operations in a generic way. It is then up to the scientist to configure the fragments according to the specific experiment. In this approach only a modeling tool is extended and hence the portability of the workflows across workflow engines is preserved. The main drawback of the approach is that selected data is always accessed by value (which can be very huge in scientific computations) and hence is processed by the engine. Moreover, scientists could end up in scripting complex XPath and XSLT expressions when configuring workflow fragments to their needs.

Another option for accessing and processing external data is by extending BPEL with particular activities. A prominent implementation of such a concept is ii4BPEL or BPEL/SQL [18], respectively. `SQL` activities execute SQL statements on databases. Thus, select, update, insert, delete, and filter data operations can be used on workflow level very similar to GriCoL. As before, the iteration would be realized by a `forEach` activity. In contrast to GriCoL, query results stay in the database's address space and are referenced by so-called `Set Reference` variables. This is helpful to keep data traffic low if data sets are not needed for steering the workflow logic. However, a `Retrieve Set` activity can be used in order to explicitly load results into the workflow engine's address space. In this case, the results are stored in specific `Set` variables. We advocate pursuing BPEL/SQL instead of the standard BPEL approach due to the multitude of benefits. The major advantage of BPEL/SQL is the possibility to automatically optimize the database accesses in BPEL workflows [19] and hence speed up workflow processing. Speedup of workflow execution can also be achieved by handling database data by reference. Furthermore, SQL activities enable a straightforward database access without complex assign activities and XPath/XSLT expressions. The main drawback of the approach is that BPEL/SQL requires an extension of the run time environment and hence impedes portability of workflows across different workflow engines. Moreover, only data of relational databases can be handled. Files or sensor data, for instance, are not accessible. Furthermore, scientists need SQL knowledge in order to specify data operations, unless there are pre-modelled queries provided to users by an advanced experiments modeling tool. To overcome these deficiencies, we are currently working on a BPEL modeling tool that contains a graphical editor to assist scientists in creating SQL statements. Another aspect of the work is a concept that allows accessing several types of external data sources besides databases (e.g. files in different formats, sensor databases).

4.2 Pipeline processing

For GriCoL pipeline connections control is transferred to the next block each time the computation of an individual data set has been completed. So, control can be transferred many times if many data sets are processed. In some cases, BPEL's `forEach` can achieve the same semantics, namely if several independent data items can be processed in parallel. However, in order to allow general pipeline semantics in BPEL an extension is needed. In [20] detailed description is given how to extend traditional state-based workflow management techniques with the necessary features to integrate streaming data services and combine them with conventional request-response services. Additionally, safety problems in the execution of the process and their solutions are addressed as state-based workflow execution models are not designed for such parallel processing. Similarly to the presented implementation for the JOpera workflow system in [20], BPEL can be extended for the support of pipelining.

For the mapping of GriCoL to BPEL it is important to

note that a BPEL `scope` activity corresponding to a GriCoL control block that can be instantiated multiple times in a pipeline must include a solver module representation that uses a standard Web service instead of a stream data-aware Web service.

4.3 Layers of abstraction / explicit data flow

As mentioned, GriCoL makes use of two layers for the specification of workflows. Solver and condition blocks are single units on the control flow layer. The data flow layer satisfies two functions. First, it is used to specify the concrete behaviour of solver and condition blocks by modules that are hidden on the control flow layer. That means the control flow layer works as abstraction from the data flow layer. Second, the data flow layer can be used to model data dependencies between blocks and modules.

Table 4: Conceptual mapping of GriCoL and BPEL concepts in order to represent two different layers of abstraction.

	GriCoL	BPEL
Control Flow Level	Basic Solver Block	Scope
	Condition control block & serial connection	Acyclic: if, flow & links Cyclic: while, repeatUntil, forEach
	Serial Connection	Sequence, flow & links
	Pipeline Connection	BPEL Extension for pipeline
Data Flow Level	Simple data selection	BPEL-D data link
	Simple data transfer	BPEL-D data link
	Data transfer	BPEL-D data link

Conceptually, a standard BPEL process model qualifies to represent a GriCoL workflow on the control flow layer (except for the pipeline mechanism that requires a BPEL extension, see Section 4.2). In order to realize the different abstraction layers for a solver block, a BPEL `scope` activity can be used as container representing a solver block on the control flow layer of GriCoL. To specify the `scope`'s behaviour particular activities have to be inserted similar to GriCoL's data flow layer. In contrast to this, different abstraction layers for a condition block cannot be implemented by BPEL concepts. However, similar behaviour can be achieved in BPEL (1) by an `if` activity or a `flow` activity with two links with mutual excluding expressions in case of an acyclic GriCoL condition block; (2) by a `repeatUntil`, `forEach`, or `while` loop in case of a cyclic GriCoL condition block (see Table 4). The data dependencies, data preparation and post-processing functionality of a condition block on the data flow layer can be mapped on appropriate BPEL behaviour in each path of the utilized BPEL concept. However, we do not advocate a BPEL extension that would realize an abstraction layer for condition block representations. We rather recommend a modeling tool extension that provides different views on workflow models highlighting different aspects.

In standard BPEL, there is no counterpart for GriCoL's capability to model data dependencies on the data flow layer. This requires explicit information about the data source and target for module representations. In BPEL, data flow is only implicitly specified with the help of variables to propagate data between activities. In theory, this concept is sufficient to express data dependencies. In practice, scientists want to model data flow explicitly. Therefore, we need a BPEL extension that accommodates a concept for explicit data links. We propose to adopt the BPEL-D extension [21] which replaces variables by explicit data links in BPEL 1.1. Since BPEL-D is a design time extension, workflow models can be transformed into a standard BPEL representation to be executed on a standard BPEL engine.

4.4 Data repository

The data repository sub-layer in GriCoL is an abstraction from concrete data sources. BPEL's variables seem to be a similar concept. Variables have a data type and are filled during workflow execution either by literal assign operations or WS invocations. In the latter case, the source of variable values is transparent for the workflow because the implementation of WSs is unknown to the workflow. The main difference to GriCoL's data repository is that BPEL variables cannot be used among different workflows.

In [22] a BPEL design time extension is proposed that allows accessing data by reference. A new type of variables is introduced that holds pointers to external data. A reference resolution system (RRS) can be used to load the value of a reference into the workflow. Loading of values can be configured to periodically, on usage, on workflow instantiation, and on external events. The RRS implements mechanisms to access data in its concrete storage location (e.g. in a database or file). Naturally, the concept of references to external data allows different workflow instances to access the same data items. The RRSs can thus be seen as a decentralized data repository.

The advantage of the approach is that only the modeling tool needs to be extended while the BPEL engine is not affected. A transformation component can translate the extended BPEL code into standard BPEL that is eventually executed. Additionally, references in BPEL are useful to keep huge amounts of data out of the engine if they are not needed for the workflow logic. This is a common situation in simulation workflows where (intermediary) results are passed between different computing resources (e.g. solvers, visualization tools) without influencing the workflow path. The downside of the solution is the need for additional components in the software infrastructure, namely the RRS and the transformation component. We are currently working on a first prototype of the reference passing mechanism.

5. Conclusion

In this paper, we argue that it is reasonable to rely on the general-purpose language BPEL to design executable simulation workflows. We therefore investigated BPEL's capabilities to express simulation workflows by

comparison to the field-tested simulation workflow language GriCoL. The comparison revealed a number of unaddressed concepts in the core BPEL specification that are critical for enabling scientific experiments and simulations. In particular, we identified that the concepts of data handling on workflow level, pipelining, different layers of abstraction/explicit data flow, and shared data would increase BPEL's expressiveness for simulation workflows. We therefore discussed approaches to extend BPEL in order to meet these specific requirements on a simulation workflow language. Since we want to pursue an engineering approach, we relied on existing extensions as far as possible. Anyhow, these existing approaches will need thorough extensions to fit into a holistic solution. For example, BPEL/SQL needs a modification to be applicable for arbitrary data sources; BPEL-D must be migrated to BPEL 2.0.

The research is one of the steps to make the conventional workflow technology more interesting, expressive and applicable for scientists to specify their simulations with workflows. In future, we want to develop a prototype that implements the findings of this work and that will be used to realize selected simulations in SimTech with workflow technology and BPEL, e.g. simulations of solidness of metals, growth of bones, or risk assessment of C02 ground storage.

Acknowledgements

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

References

[1] Taylor et al., *Workflows for e-science – Scientific workflows for grids* (Springer, 2007).

[2] Currle-Linde et al., GriCoL: A language for scientific grids, *Proc. 2nd IEEE International Conf. on e-Science and Grid Computing*, 2006

[3] Leymann & Roller, *Production workflows: Concepts and techniques* (Prentice-Hall, 2000).

[4] OASIS, Web services business process execution language (BPEL) version 2.0, OASIS Standard, April 11th, 2007. [online] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

[5] Currle-Linde et al., Science experimental grid laboratory (SEGL) – Dynamical parameter study in distributed systems, *International Conf. on Parallel Computing*, 2005.

[6] Weerawarana et al., *Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More* (Prentice Hall, 2005).

[7] van Lessen et al., An execution engine for semantic business processes, *Proc. of Service-Oriented Computing – ICSOC Workshop*, 2007

[8] Khalaf et al., Pluggable framework for enabling the execution of extended BPEL behavior, *Proc. of the 3rd International Workshop on Engineering Service-Oriented Application (WESOA)*, 2007.

[9] IBM: "WebSphere Process Server". [Online] `http://www-01.ibm.com/software/integration/wps/`

[10] Oracle: "Oracle BPEL Process Manager". [Online], `http://www.oracle.com/appserver/bpel_home.html`

[11] Bouziane et al., A software component-based description of the SEGL runtime architecture, *Proc.* of the CoreGRID Integration Workshop, pp. 261-273, Krakow, Poland, 2006.

[12] Kloppmann et al., WS-BPEL extension for sub-processes – BPEL-SPE, White Paper, 2005.

[13] Agrawal et al., WS-BPEL extension for people (BPEL4People), Version 1.0, White Paper, 2007.

[14] Altintas et al., Kepler: An extensible system for design and execution of scientific workflows. *Proc. of the International Conf. on Scientific and Statistical Database Management*, 2004.

[15] W3C, XML Path Language (XPath) 2.0, W3C Recommendation, January 23rd, 2007. [online] `http://www.w3.org/TR/xpath20`

[16] W3C, XSL Transformations (XSLT) Version 2.0, W3C Recommendation, January 23rd, 2007. [online] `http://www.w3.org/TR/xslt20`

[17] Eberle et al., Process fragments, In: Meersman et al. (Eds.), *On the Move to Meaningful Internet Systems (OTM) Part I*, 2009.

[18] Vrhovnik et al., An overview of SQL support in workflow products, *Proc. of the 24th International Conf. on Data Engineering (ICDE 2008)*, Cancún, México, April 7-12, 2008.

[19] Vrhovnik et al., PGM/F: A framework for the optimization of data processing in business processes, *Proc. of the 24th International Conf. on Data Engineering (ICDE 2008)*, Cancún, México, April 7-12, 2008.

[20] B.J. Biörnstad, A workflow approach to stream processing, PhD thesis, ETH Zurich, 2008.

[21] Khalaf & Leymann, Role-based decomposition of business processes using BPEL, In: *Proc. International Conf. on Web Services (ICWS)*, 2006, 770-780.

[22] Wieland et al., Towards reference passing in web service and workflow-based applications, *Proc. of the 13th IEEE Enterprise Distributed Object Conference (EDOC)*, 2009.

[23] Barga & Gannon, Scientific versus business workflows, In: Taylor et al. (Eds.), *Workflows for e-science – Scientific workflows for grids* (Springer, 2007).

[24] Wassermann et al., Sedna: A BPEL-based environment for visual scientific workflow modeling, In: Taylor et al. (Eds.), *Workflows for e-science – Scientific workflows for grids* (Springer, 2007).

[25] Akram et al., Evaluation of BPEL to scientific workflows, *Proc. of 6th IEEE International Symposium on Cluster Computing and the Grid*, 2006.