# Handbook of Research on Architectural Trends in Service-Driven Computing

Raja Ramanathan
*Independent Researcher, USA*

Kirtana Raja
*IBM, USA*

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Information Science REFERENCE
An Imprint of IGI Global

# Chapter 3
# Distributed and Adaptive Business Process Execution: A Scalable and Performant Solution Architecture

**Michael Pantazoglou**
*National and Kapodistrian University of Athens, Greece*

**Aphrodite Tsalgatidou**
*National and Kapodistrian University of Athens, Greece*

**George Athanasopoulos**
*National and Kapodistrian University of Athens, Greece*

**Pigi Kouki**
*National and Kapodistrian University of Athens, Greece*

## ABSTRACT

*Centralized business process execution engines are not adequate to guarantee smooth process execution in the presence of multiple, concurrent, long-running process instances exchanging voluminous data. In the centralized architecture of most BPEL engine solutions, the execution of BPEL processes is performed in a closed runtime environment where process instances are isolated from each other, as well as from any other potential sources of information. This prevents processes from finding relative data at runtime to adapt their behavior in a dynamic manner. The goal of this chapter is to present a solution for the performance improvement of BPEL engines by using a distributed architecture that enables the scalable execution of service-oriented processes, while also supporting their data-driven adaptation. The authors propose a decentralized BPEL engine architecture using a hypercube peer-to-peer topology with data-driven adaptation capabilities that incorporates Artificial Intelligence (AI) planning and context-aware computing techniques to support the discovery of process execution paths at deployment time and improve the overall throughput of the execution infrastructure. The proposed solution is part of the runtime infrastructure that was developed for the environmental science industry to support the efficient execution and monitoring of service-oriented environmental science models.*

DOI: 10.4018/978-1-4666-6178-3.ch003

## INTRODUCTION

Alongside high-level business process notation languages such as BPMN 2.0 (OMG, 2011), Web Services Business Process Execution Language (Alves et al., 2007), abbreviated to WS-BPEL or BPEL, is widely considered to be the de facto standard for the implementation of executable service-oriented business processes as compositions of Web services.

In many cases, which have become evident in various application domains, centralized BPEL engines are clearly not adequate to guarantee smooth process execution, and thereby ensure client satisfaction in the presence of multiple, concurrent, long-running process instances exchanging voluminous data. Indeed, as the numbers of clients grow, the underlying infrastructure needs to maintain and handle multiple process instances while waiting for the external Web services that are invoked to complete their execution.

In some cases, *clustering* techniques can be employed to address the scalability issue, by dispatching the execution of each incoming process request to the BPEL engine residing on the cluster member with the lowest workload. However, the deployment and maintenance of clusters consisting of two or more centralized BPEL engines, sets requirements on the underlying hardware resources that cannot be always fulfilled by the involved organizations. Furthermore, clustering could prove to be an inefficient approach under certain conditions, as it cannot overcome the emergence of bottlenecks that are caused by specific activities of a BPEL process. Moreover, as the execution of a process instance still takes place in a centralized manner, issues relating to large volumes of data are not effectively addressed. In such context, inevitably, the BPEL engine becomes bloated with pending requests coming from multiple concurrent clients. Hence, the overall throughput of the execution infrastructure is dramatically deteriorated, while the process execution times escalate to unacceptable levels.

Aside from the aforementioned scalability issues that derive from the *centralized architecture* of most BPEL engine solutions, the execution of BPEL processes is also performed in a closed runtime environment. More specifically, process instances are isolated from each other, as well as from any other potential sources of information. This prevents processes from finding and exploiting relative data at runtime, in order to improve their predefined behavior in a dynamic manner. By relative data we refer to semantically annotated, structured data that are semantically associated to a given process. Instead, it becomes the responsibility of the process designer to manually adapt the process specification so as to accommodate emerging data sources. For example, rendering a weather calculation process able to incorporate data stemming from a satellite that was not available during process design-time would deem process redesign.

In order to address all these challenges, we propose a *decentralized BPEL engine architecture* with data-driven adaptation capabilities. Our engine employs the *hypercube peer-to-peer (P2P)* topology along with a set of distributed algorithms in order to improve the average process execution times, and the enhancement of the overall throughput of the execution infrastructure in the presence of multiple, concurrent, and long-running process instances.

In addition to the decentralized architecture, the proposed engine accommodates the provisioning of adaptable BPEL processes by exploiting information available to the process environment along with existing services. Adaptation in the context of our approach is about the identification and use of possible alternatives for the achievement of the goals and sub-goals defined in a BPEL process; these include the utilization of available, related information and/or services (or service chains).

*Data-driven adaptation* incorporates Artificial Intelligence (AI) planning and context-aware computing techniques to support the discovery of process execution path substitutions at deployment time. When calculating the possible choices, the goal of our approach is to reduce the number of steps, i.e., the number of activities defined in the original process. In the context of our approach we argue that the reduction of unnecessary process activities can lead to shorter execution times. In this way, data-driven adaptation complements the enhancement of the overall performance of our decentralized BPEL engine.

The chapter is organized as follows: the background section provides an overview of the literature in decentralized BPEL process execution and data-driven adaptation and includes a case study that highlights the need for adaptation as well as improving the performance of service oriented processes. This is followed by a detailed presentation of the architecture solution for improving the performance of service-oriented process executions based on the use of distributed and adaptable processes, by using an illustrative example from the environmental science domain. Finally, future research directions are discussed and the conclusion is presented.

## BACKGROUND

*WS-BPEL 2.0* is a widely known OASIS standard used for the provisioning of executable business processes. Since almost the onset of the Service-oriented Computing vision, WS-BPEL has emerged as the prominent approach for the interoperable specification of intra-corporate and business-to-business interactions, by providing a model and a grammar capable of describing the behavior of a business process in terms of interactions between the process and its partners. As it is a popular language for the specification of service-oriented processes, it is briefly presented and analyzed here.

Technically, WS-BPEL provides a language for the formal specification and modeling of both forms of business processes: executable and abstract business processes. Executable business processes model actual behavior of a participant in a business interaction, whereas abstract business processes use process descriptions to specify the mutually visible message exchange behavior of the parties involved in the process. A brief list of its principal characteristics is the following:

- It is an XML-based language that is based on XML Schema (Sperberg-McQueen & Thompson, 2000) for the definition of data structures and on XPath (Clark & DeRoso, 1999) for retrieval of XML elements (data manipulation).
- It models a business process as a composition of elementary Web Services and depends on the W3C standards WSDL (Christensen et al., 2001) for the description of the inputs, outputs, and operations of a Web service.
- WS-BPEL defined business processes are exposed as Web services (WSs), so they can be invoked from another business process.

A WS-BPEL process specification is defined in terms of the following six main artifacts.

- **Partner Link:** The interaction with each partner occurs through Web service interfaces. Particularly, a Partner Link encapsulates the structure of the relationship at the interface level between two partners (e.g., a Web Service and a process). A respective partner link type must be first defined to specify the required and provided WSDL port types. As we will see below, while partner link is the one which provides the communication channel to remote WSs, the use of partner link type creates problems.
- **Variables:** They are used to store both message data of Web service interactions and control data of the process.
- **Correlation:** Correlation sets specify in which business process instance a returned

message from a WS should be retrieved, and that because long-running business processes are supported, so there may be several process instances waiting for the arrival of a Web service message.

- **Basic Activities:** They are separated in activities that communicate with Web services (invoke, receive, reply), in activities that manipulate variables (assign), and in activities that wait or terminate a process (wait, exit).
- **Structured Activities:** They can define the control flow of basic activities. They include activities which specify sequential/parallel execution (sequence/flow), activities that decide which branch will be executed (if-else), and activity loops (while).
- **Handlers:** They are provided so as to deal with unexpected or exceptional situations and they are available in two forms, event handlers and fault handlers.

In general, a workflow (or seamlessly a process model), consists of three dimensions:

1. Process logic, namely "what" is to be done?
2. Organization, namely "who" does it?
3. Infrastructure, namely "which" tools are used?

In WS-BPEL the "what" dimension is based on activities and the "which" dimension is based on Web Services. From the moment that activities directly refer to WSDL operations in order to call a Web Service, we infer that "which" and "what" dimensions are closely related. Indeed this bond is far from desirable and has been the source of severe criticism on WS-BPEL.

This strong bond hinders the exploitation of services, which do not comply with the WSDL specification. Thus, the flexibility and reusability properties of WS-BPEL are largely affected. The advent of *Semantic Web* (McGuinness & Harmelen, 2004) and *Semantic Web Services* (SWS) (Steinmetz & Toma, 2008; Martin et al., 2004) has aggravated this problem. SWS provide a declarative description of the service functionality, contrary to conventional Web Services where syntactic descriptions are the prime means for service interface definition. SWS give the opportunity to be discovered by criteria based on their functionality and not on their signature. This new opportunity cannot be directly exploited by WS-BPEL due to its strong bond with WSDL.

Another strong point of criticism to WS-BPEL is its strict nature, which poses significant barriers in the provisioning of dynamic process models. As business models (and process models consequently) mature, the ability to evolve and adapt to changing conditions is becoming a necessity. Process models defined in WS-BPEL are unable to accommodate changes in user requirements and operational environments due to the inherently static nature of the WS-BPEL process flow specification. Thus, a process defined in WS-BPEL has to be redesigned in cases where additional services or information have to be integrated. This inability is a significant barrier to the use of WS-BPEL in the provisioning of modern context-aware systems and many approaches have emerged in order to surpass it.

## Distributed Scientific Workflow Systems

In the last few years, developments in *Scientific Workflow* (SWF) systems have made possible the efficient and scalable execution of long-running workflows that comprise large numbers of parallel tasks; and operate on large sets of data. Since the challenges met by those efforts bear some resemblance to the motivation behind our work, we would like to emphasize on their different scope and technical foundations.

By definition, the majority of SWF solutions are particularly designed to support the modeling and execution of in silico simulations and scientific experiments. Moreover, they are mostly based on proprietary executable languages, which are tailor-made to the needs of such applications. On the other hand, by using the BPEL standard as its underlying basis, our engine has more general purposes and can be used to support a wider range of environments and applications. The different scopes of our proposed engine and the various SWF systems are also reflected by their pursued programming models.

Due to their data-flow orientation, most SWF engines, for instance Taverna (Missier et al., 2010), follow a functional data-driven model, whereas BPEL engines, including the one presented in this chapter, implement an imperative control-driven model. Hence, the focus of our contribution is on implementing algorithms that distribute the control flow of processes, in a way that no central coordinator is required. On the other hand, since the control flow is of minor importance to scientific workflows, SWF systems build on efficient parallelization and pipelining techniques, in order to improve the processing of large-scale data flows.

For instance, Pegasus (Callaghan et al., 2010) attains scalability by mainly addressing the large number of parallel tasks in a single workflow, and the corresponding voluminous data sets, through task clustering and in-advance resource provisioning. In our work, we are primarily interested in improving the throughput of the BPEL engine, defined as the number of incoming process requests being served per minute, and the average process execution times, in the presence of large numbers of concurrently running process instances that are long-running and consume potentially large data sets.

Most SWF systems, e.g., Kepler (Altintas et al., 2004), Triana (Taylor et al., 2003), or Pegasus (Deelman et al., 2005) exhibit a clear separation of concerns between the design of a workflow and the execution infrastructure, although much effort has been spent on supporting Grid settings such as Globus. In general, however, Grid infrastructures are heavyweight, complex, and thus difficult to manage and maintain. In contrast, our BPEL engine is able to seamlessly organize and manage any set of nodes in a hypercube topology, so as to engage them in the execution of long-running and resource-demanding processes.

Still, despite their inherently different scopes, programming models, and scalability concerns, SWF systems have effectively dealt with advanced data management aspects, such as provenance (Altintas, Barney, & Jaeger-Frank, 2006), or high-speed data transfer (Allcock et al., 2005). These features are complementary to our approach and could be accommodated by our BPEL engine to further enhance its capabilities and performance.

## BPEL Decentralization

The decomposition and decentralized enactment of BPEL processes is a valid problem that has been the subject of many research efforts in the past years. In this section, we review a number of related results that have become available in the literature.

A P2P-based workflow management system called SwinDeW that enables the decentralized execution of workflows was proposed by Yan, Yang, & Raikundalia (2006). According to the authors, the generic workflow representation model is compatible with most concrete workflow languages including BPEL, although this compatibility is not demonstrated. In any case, similar to our presented approach, SwinDeW is based on the decomposition of a given workflow into its constituent tasks, and their subsequent assignment to the available nodes of a P2P network, in order to remove the performance bottleneck of centralized engines.

A main difference between that approach and the one presented in this chapter lies in their corresponding worker recruitment algorithms: SwinDeW makes use of the JXTA (Gong, 2001) peer discovery mechanism to find nodes with specific capabilities, and then quantifies their workload before assigning the given task to the one with the minimum workload. Since the respective discovery protocol cannot guarantee that all relevant peers will be found upon a query, it may become possible that not all available nodes in the P2P network are equally utilized. In our approach, the recruitment algorithm relies on the hypercube topology, the inherent ability to perform efficient random walks, and the frequency of use of each node in order to evenly divide the workload and thereby exploit all available resources.

The NINOS orchestration architecture (Li, Muthusamy, & Jacobsen, 2010) is based on a distributed content-based publish/subscribe (pub/sub hereinafter) infrastructure, which is leveraged to transform BPEL processes into fine-grained pub/sub agents. The latter then interact using pub/sub messages and collaborate in order to execute a process in a distributed manner. A critical departure of our work from the NINOS approach lies in the respective process deployment mechanisms. In NINOS, a BPEL process is deployed prior to its execution on a number of agents, which remain the same for all subsequent executions of the process. Hence, the infrastructure may underperform in the presence of multiple concurrent instances of the same process. In our case, the BPEL process is decomposed and its constituent activities are assigned to the available nodes in the P2P network at runtime, depending on their current workload, which is inferred by their frequency of use.

In an attempt to improve the throughput of the BPEL process execution infrastructure in the presence of multiple concurrent clients, a program partitioning technique has been proposed by Nanda, Chandra, & Sarkar (2004), which splits a given BPEL process specification into an equivalent set of processes. The latter are then executed by different server nodes without the need of a centralized coordinator. Similar approaches have also been proposed by Baresi, Maurino, & Modafferi (2006) as well as by Yildiz & Godart (2007). Along the same lines, the use of a penalty-based genetic algorithm to partition a given BPEL process and thereby allow decentralized execution was proposed by Ai, Tang, & Fidge (2011).

However, to realize these partitioning techniques, each participating node must host a full-fledged BPEL engine, which is often heavyweight and therefore not always affordable by many small organizations and businesses. In our approach, there is no such requirement imposed on the nodes forming the underlying P2P infrastructure, and thus each node has a relatively small memory footprint. This way, our distributed BPEL engine can leverage and be deployed on hardware with limited capabilities.

A solution to the problem of decentralized BPEL workflow enactment that is based on the use of tuplespace technology was reported by Wutke, Martin, & Leymann (2008). According to that approach, workflows are defined as BPEL processes, which are split among all participating partners, and are implemented directly by the individual components. The latter are deployed and coordinate themselves using shared tuplespace(s). Like our approach, the tuplespace technology facilitates the execution of data-intensive workflows, since it allows for data distribution and yields a decrease of messages being passed between the interacting components. Unlike our approach, however, the overall decomposition requires considerable preparatory work such as component configuration to be conducted at deployment time, which could eventually become a scalability bottleneck.

In order to effectively separate the concerns of regular BPEL engines and various other complex aspects, including decentralized orchestration, Jimenez-Peris, Patino Martinez, & Martel-Jordan (2008) proposed the ZenFlow BPEL engine. ZenFlow employs techniques from reflective and aspect-oriented programming, and makes use of specialized meta-objects to describe and plug the extensions necessary for de-centralized execution into the centralized BPEL engine. In this work, however, decentralization is enabled by means of a cluster of centralized BPEL engines, with each one being responsible for the execution of the whole process each time. We follow a fine-grained decentralization strategy, whereby the

BPEL process is decomposed into the constituent activities, the execution of which is distributed among the nodes of a P2P network.

The CEKK machine that was presented by Yu (2007) supports P2P execution of BPEL processes based on the continuation-passing style. In this approach, the execution control is formalized as a continuation, and is passed along from one processing unit to another without the interference of a central coordinating component. In this distributed execution environment, special attention is paid to the problem of failure handling and recovery, while a number of extensions to the BPEL language are introduced. Overall, this approach focuses on the formalization of a distributed process execution model and does not address aspects related to the structure of the P2P infrastructure, or the distribution of process activities and variables. Furthermore, it lacks an evaluation that would allow us to assess its applicability to the execution of long running and data-intensive BPEL processes.

Details of the proposed architecture are described in the following section.

## ADAPTIVE AND SCALABLE PROCESS EXECUTION ARCHITECTURE

This section first presents a motivation case study followed by the architecture of the proposed Adaptive Execution Infrastructure and describes its main components and functionality.

### Case Study

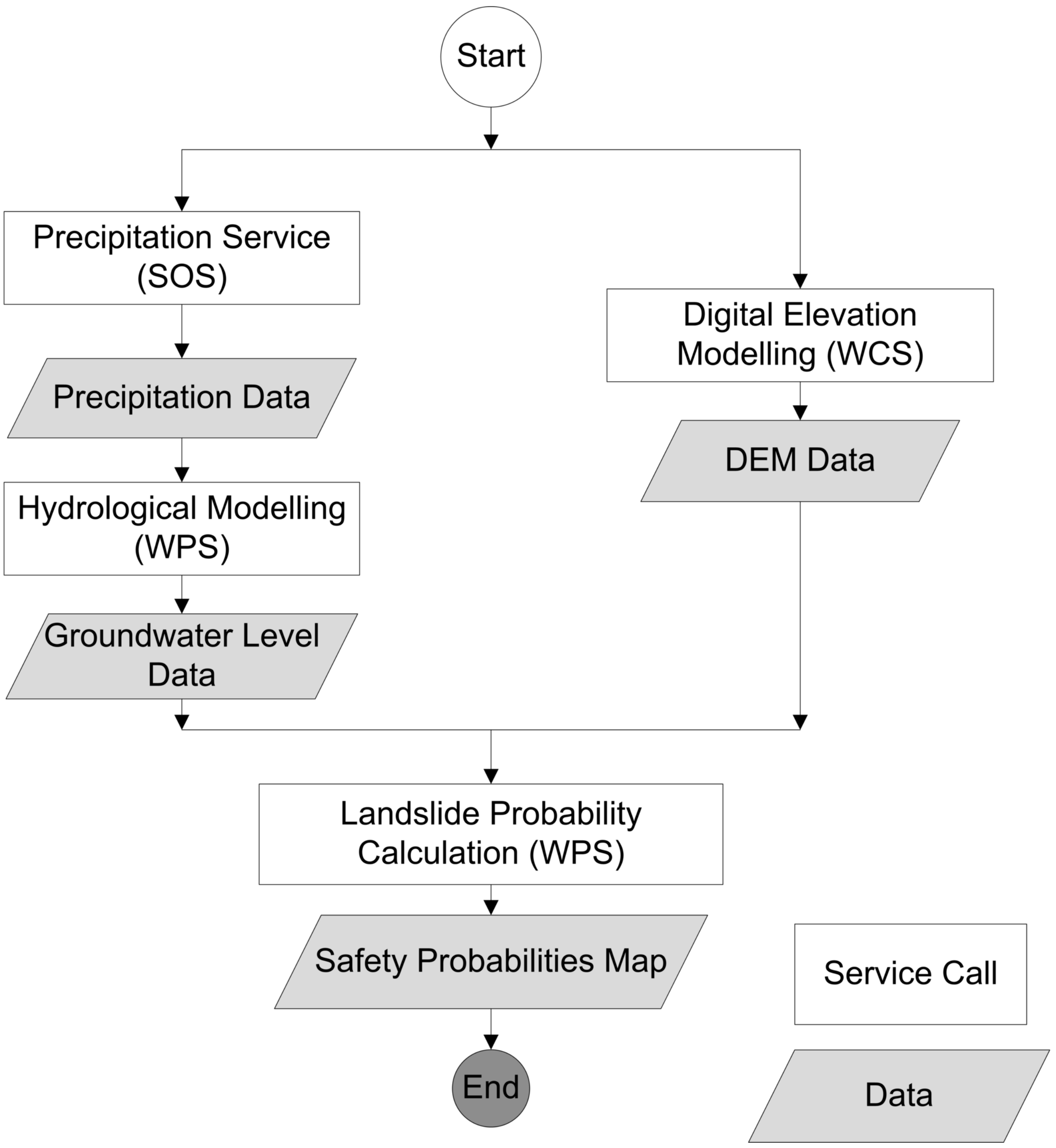
To better illustrate the motivation and need for improvements on the performance of service-oriented processes, we hereby present a case study, which stems from the environmental domain. More specifically, the presented case study is related to the estimation of the landslide probability at a given area in Guadeloupe. The illustrated process was designed and developed in the ENVISION (ENVIronmental Services Infrastructure with Ontologies) project[1] and is part of the decision support system dedicated to landslide risk assessment.

As it can be seen in Figure 1, the landslide process orchestrates four OGC[2] (Open Geospatial Consortium) Web Services. First, a digital elevation model of the specified area is retrieved by invoking a *Web Coverage Service (WCS)* (OGC, 2012) through activity *Digital Elevation Modeling*.

In parallel, a *Sensor Observation Service (SOS)* (OGC, 2007a), named Precipitation Service, is called in order to retrieve the precipitation data of the user-specified area. This data along with a set of user input parameters are fed to a *Web Processing Service (WPS)* (OGC, 2007b) titled Hydrological Modeling, which simulates the main mechanisms of the water cycle by a system of reservoir. The produced digital elevation model and the hydrological model containing the produced map of groundwater level in that area are finally passed as input to another WPS called Landslide Probability Calculation, which performs static mechanical analysis in order to calculate the landslide probabilities in the area of study, in the form of a map of safety factors ranging between zero and one. That map is finally returned to the user as the process output.

Even though the presented process model (Figure 1) is rather simple in terms of control flow, the complexities of the incorporated operations, which are implemented by the specified external services, render it a long-running process. Moreover, considering that this is usually executed repeatedly, its performance becomes an issue of paramount importance. In addition, another requirement that is clearly highlighted is the need to provide several customizations to the executing process in order to simulate the distinct conditions of several execution scenarios. For example, process clients would like to use historical measurements of ground water cycles, and digital

*Figure 1. The Landslide process model*


Start
Precipitation Service (SOS)
Precipitation Data
Hydrological Modelling (WPS)
Groundwater Level Data
Digital Elevation Modelling (WCS)
DEM Data
Landslide Probability Calculation (WPS)
Safety Probabilities Map
End
Service Call
Data

elevation models calculated from different sensors or cached measurements.

Therefore, reducing the process execution time and supporting adaptation would assist decision makers in selecting and customizing the actions that should be performed in order to compensate possible landslide effects. In this frame, contemporary languages and mechanisms used for the specification and offering of service-oriented processes cannot accommodate these challenges. More advanced solutions are clearly needed.

## Data-Driven Adaptation

The prime assumption of the *Data-Driven Process Adaptation approach* (DDA) is that a service-oriented process comprising heterogeneous services should be able to use the information available within its environment and adapt its execution accordingly. To facilitate the provisioning of such adaptable processes, Athanasopoulos & Tsalgatidou (2010), proposed an approach that exploits information contained within a specific "space" to adapt a service-oriented process.

The space is considered to be the process's environment, which is open to other processes and systems for information exchange. Appropriate algorithms specify adaptation paths for given processes along with queries that can be executed in the shared space; these queries search for relevant information, which when found, is fed to a process execution engine. The execution engine uses the discovered information for controlling the execution and adaptation of running process instances according to the adaptation paths specified by the provided algorithms.

The proposed solution accommodates the necessary components to address the following three basic functional needs: collection of contextual information, execution of heterogeneous service processes, and process adaptation driven by collected information. Therefore, the accommodating infrastructure comprises three main components:

- A *Semantic Context Space Engine* (SCS Engine) that supports the exchange of contextual information.
- A *Service Orchestration Engine* that executes heterogeneous service processes and uses contextual information to adapt a running process.
- A *Process Optimizer*, which generates Data Driven Adaptable Service-Oriented Processes (DDA-SoP).

The SCS Engine provides an open space where one may i) write and retrieve information, which is annotated with meta-information, and ii) logically group information of interest, e.g., information pertaining to a specific domain, such as, weather conditions, and specify associations among groups, which contain information from related/depending domains, e.g., a weather conditions group can be associated to a group with information on the aquatic conditions of a specific region.

The Service Orchestration Engine provides a BPEL-based engine executing heterogeneous service orchestrations, e.g., comprising Web, Grid, P2P, and OGC services (Doyle et al., 2001). The orchestration engine supports the monitoring and reconfiguration of running process instances according to the suggestions made by the Process Optimizer.

The Process Optimizer component implements an AI planner to discover process plans controlling the execution and adaptation of service processes upon the emergence of related information. Specifically, according to Athanasopoulos & Tsalgatidou (2010), the problem of Data-Driven Adaptation can be modeled as a non-deterministic, partially observable planning problem (Ghallab, Nau, & Traverso, 2004). Indeed, considering that services used in a process model act as black boxes, whose outcome can vary e.g., both normal and abnormal outcomes in case of system failures, the behavior of the process cannot be deterministically identified at each time. In this frame, solutions are modeled as conditional plans, which contain branching control structures, i.e., if-then-else, that decide on the execution path that will be followed based on the values of specified conditions.

A crucial step in the provisioning of data-driven adaptable service-oriented processes is the introduction of extensions to the planning problem representation. This step can be regarded as the incorporation of additional "sensors" for monitoring the process, along with appropriate actions for handling the accruing "observations". More specifically, this extension process comprises the following:

- The semantic-based extension of observations.
- The extension of the action set with actions capable of supporting the exploitation of the introduced observations.
- The consolidation of the extended action and observation sets.

The execution of the aforementioned expansion actions vies to support the introduction of appropriate adaptation steps that would enable the execution of alternate process paths upon the discovery of related information. In the context of the DDA approach, adaptation steps are selected points in a process model where the existence or absence of appropriate information, i.e., observations, could be exploited for deciding whether an alternate service or service chain could be used. Adaptation steps are introduced so as to reduce the set of actions that have to be executed for achieving the process goal.

## Solution Architecture

Overall, the solution is characterized by the following features.

- **Fully Decentralized, P2P-Based BPEL Engine Architecture:** BPEL processes are deployed, executed, and monitored by a set of nodes organized in a hypercube P2P topology. Each node does not fully take charge of executing the whole process; rather, it contributes by running a subset of the process activities, and maintaining a sub-set of the generated process data. Thus the BPEL execution engine is fully operational without the need of any central controller components.
- **Fine-Grained Distribution of Process Activities:** Decentralization of process execution fits to the nature of long-running business-to-business interactions, and significantly improves the performance and throughput of the execution infrastructure. BPEL processes are fully de- composed into their constituent activities. Large-scale parallelization is feasible as the various activities designated to run in parallel can be synchronized and executed by different nodes.
- **Proximity-Based Distribution of Process Variables:** Since in many application domains processes consume and produce large volumes of data, it is important that those data are distributed in order to avoid resource exhaustion situations. Our algorithms make sure that the data produced by a BPEL process will be distributed across the nodes involved in its execution. Moreover, they will stay close to the process activities that produce them, thereby avoiding the unnecessary transfer of potentially large volumes of data between nodes as much as possible.
- **Asynchronous Interaction with the Client:** Even if a BPEL process is synchronous, following the request-response communication pattern, the interaction between the client and the distributed execution engine occurs in an asynchronous, non-blocking manner. This way, the execution engine is able to serve multiple long-running process instances without the need to maintain open connections to the respective clients over long periods of time. Furthermore, while waiting for a long-running process instance to complete, clients are given the monitoring mechanisms to retrieve intermediate results, without in-

tervening or inflicting additional delays on the process execution.

- **Efficient Use of the Available Resources and Balanced Workload Distribution:** The proposed algorithms ensure that all nodes available in the P2P infrastructure will contribute to the execution of BPEL processes. The frequency of use of each node is taken into account upon load balancing, while efficient routing techniques are employed in order to achieve an even distribution of the workload at any given time and thereby avoiding the emergence of performance bottlenecks.
- **Reuse of Available Data and Services for Process Adaptation:** The provided BPEL processes are expanded at deployment time with alternative execution paths and extension points. The latter are evaluated at runtime, so as to accommodate process adaptation based on the exploitation of available information elements, which are external to the process execution context.
- **Exchange of Semantically Annotated Information with External Sources:** BPEL process instances are able to exchange information with external sources as long as the provided information is annotated with appropriate semantics. The infrastructure complies with a Linda-based architectural model; hence, it permits the interaction of BPEL processes with any type of external information source. This mechanism is extendable with regards to the metadata primitives used for the annotation of information elements and supports their logical organization into groups.
- **Calculation of Process Adaptation Paths at Pre-Execution Time:** Calculation of possible process adaptation paths involves the execution of time consuming computations and interactions with external systems, e.g., service registries. To avoid the overhead that would be introduced to the process execution by an *on-the-fly adaptation* approach, we perform all required computations at process deployment time.

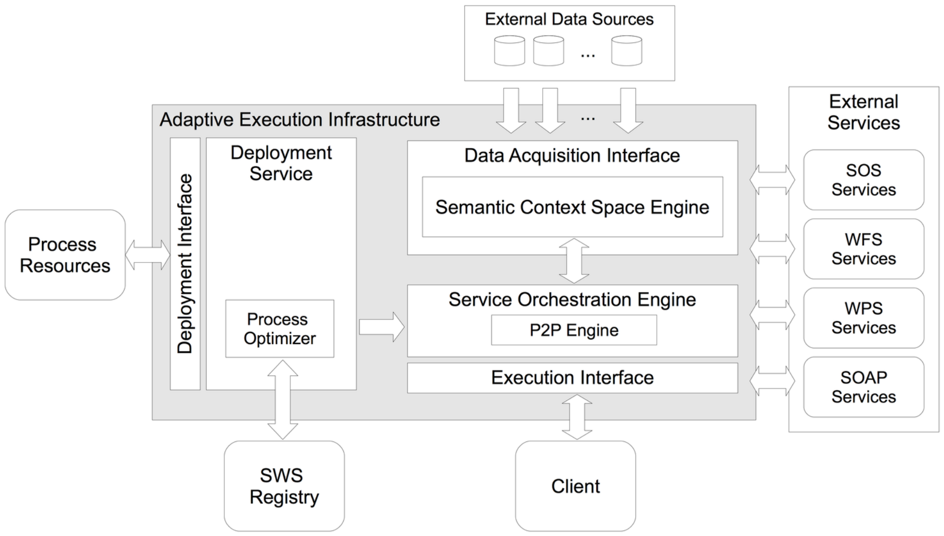
All the aforementioned features are supported by the engine in a transparent fashion. Thus, no additional overhead is imposed on the BPEL process designer, the process clients, and the administrator of the execution infrastructure. Figure 2 illustrates a high-level architectural view of the Adaptive Execution Infrastructure. As it can be seen, it comprises of a set of three main components, namely the *Deployment Service*, the *Semantic Context Space (SCS) Engine*, and the *Service Orchestration Engine*. The infrastructure also provides a set of interfaces, namely the *Deployment Interface*, the *Data Acquisition Interface*, and the *Execution Interface*. These components and interfaces enable the deployment and adaptive execution of BPEL processes in a distributed and scalable manner.

Let us briefly describe how the above mentioned components cooperate with each other during the deployment and execution of an environmental science model that has been implemented as a BPEL process.

## Deployment Service

The Deployment Service is the entry point to the Adaptive Execution Infrastructure. It supports the deployment of WS-BPEL processes (e.g., environmental science models), as well as the un-deployment of the models that were previously deployed but are no longer used, or they need to be substituted. In a typical usage scenario, this component accepts a bundle from the client, which contains the BPEL process file and all its accompanying artifacts i.e., the WSDL (Christensen, Curbera, Meredith, & Weerawarana, 2001) documents of the constituent services, the external XSD (Sperberg-McQueen & Thompson, 2000) files, and any required XSLT (Clark, 1999) files.

*Figure 2. High-level architecture of the Adaptive Execution Infrastructure*



The contents of the submitted bundle are processed by the *Process Optimizer*, which is an internal component of the Deployment Service and is further described in the remainder of this paragraph. The outcome of this processing is an expanded BPEL process definition that is dispatched to the Service Orchestration Engine. In turn, the latter binds the deployed BPEL process to a unique Web service endpoint address. Hence, in compliance with standards and common practices, all BPEL processes that are deployed to the Adaptive Execution Infrastructure can be conveniently invoked as standard SOAP Web services.

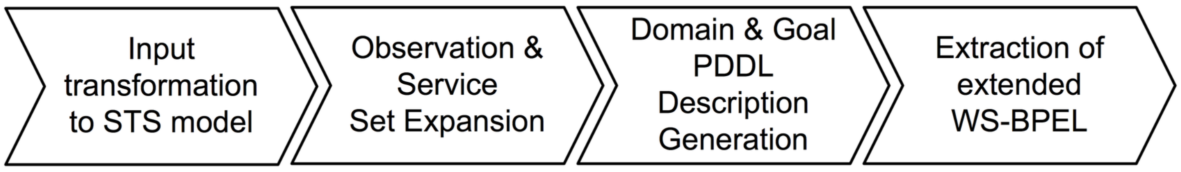
The Process Optimizer performs all necessary work to expand and render the originally submitted BPEL process adaptive. Its main objectives are to (i) expand the provided BPEL processes with extension points that are evaluated at runtime, and (ii) accommodate process adaptation based on the exploitation of available information elements, which are external to the process execution context.

These objectives are met by the Process Optimizer by specifying (i) the set of information elements which are relevant to a given process model, and should be pushed by the SCS Engine to the model instances upon execution, and (ii) the adaptation steps (equivalently referred to as *plans*) that should be performed upon the discovery of such information at runtime.

Taking a closer look into the operational semantics of the Process Optimizer, there are four distinct phases in the generation of the extended process model (see Figure 3). These distinct steps support:

- The transformation of the provided input (i.e., the bundle fed to the Deployment Service) into an internal finite state machine model representation (referred to as State Transition System model, or STS model, hereinafter).
- The expansion of the generated STS model with the inclusion of observations and additional actions (i.e., service operations).
- The generation of Planning Domain Definition Language (PDDL)-based representations for the extended planning problem domain and goal descriptions which are fed to an external Artificial Intelligence (AI) planner.
- The extraction of the extended WS-BPEL specification out of the planner outcomes.

*Figure 3. Phases of BPEL process expansion*


Input transformation to STS model
Observation & Service Set Expansion
Domain & Goal PDDL Description Generation
Extraction of extended WS-BPEL

The above steps are implemented by the internal components of the Process Optimizer, which are shown in Figure 4. More specifically, the *Input Translator* is responsible for managing the transformation of the provided input to a finite state machine model (STS model), whilst its comprising sub-components, namely the *Observation & Service Expansion Engine* and the *Planner Input Producer*, are responsible for the expansion of the generated STS model, and the planning problem domain and goal descriptions, respectively.

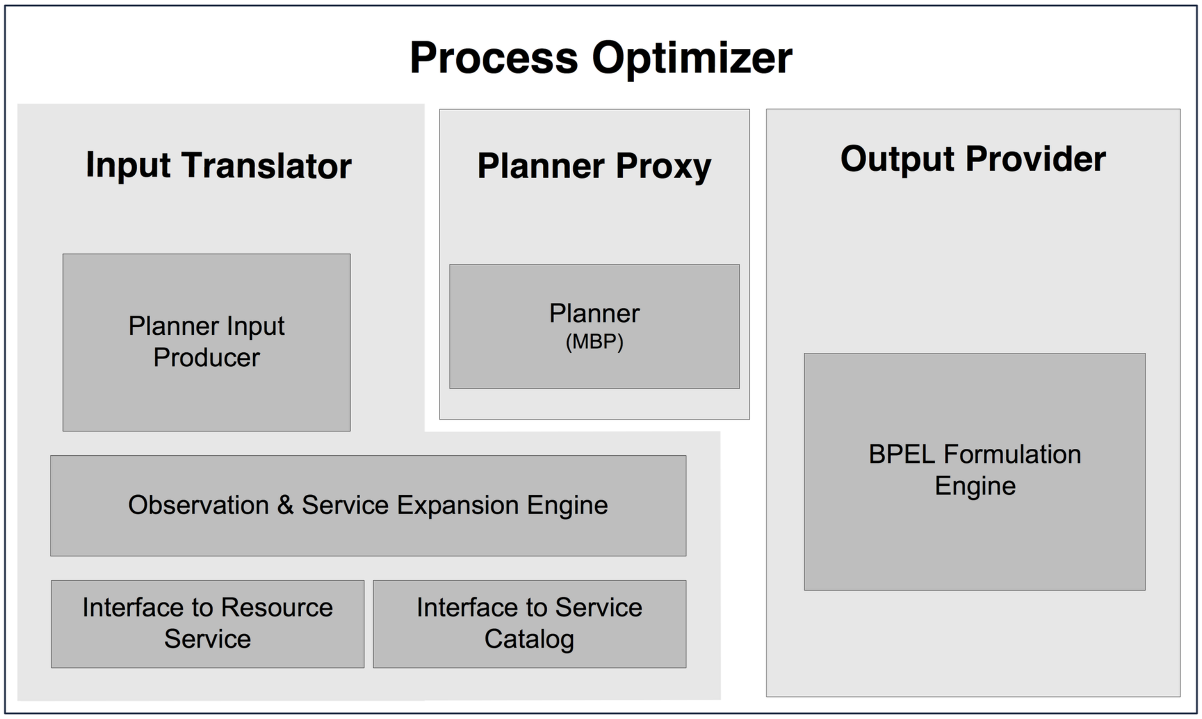
The planning problem domain and goal descriptions provide an abstract representation of the set of available activities (along with complementary descriptions of variables, constants, states, transitions, etc.) and of the expected initial and final states of the requested controlling automaton (i.e., the STS representation of the extended process). We need to state here that final states (or goal states) of a process model can be easily identified as these correspond to the resulting states of the normal or abnormal, i.e., exceptional, process ends; these are also semantically annotated as they correspond to the exposed process service outcomes, which are described in the ushering semantic description. This abstract representation is described in terms of Non-deterministic Planning Domain Definition Language (NuPDDL) (Bertoli et al., 2003) constructs. The *Planner Proxy* component provides a wrapper service to the employed planner, and the *Output Provider* facilitates the extraction of the extended BPEL descriptions based on the outcome of the planner.

The execution flow along with the artifacts exchanged between the components of the Process Optimizer and the external components are presented in Figure 5. As it can be seen, the provided process specification (i.e., the BPEL file) and the associated service descriptions (i.e., the WSDL documents) are all fed to the Input Translator. The Input Translator generates the corresponding STS model representations using, in addition to the provided BPEL and WSDL descriptions, the semantic descriptions of all related services i.e., WSMO-Lite specifications (Vitvar et al., 2008).

All these artifacts are then pushed to the Observation & Service Expansion Engine, which defines an initial set of observations on the given process model and then expands it based on the use of appropriate semantic similarity measures. The Observation & Service Expansion Engine is then able to identify additional services, which are also transformed to STS representations. All the STS representations are combined by the Planner Input Producer and jointly constitute the planning problem domain. The planning problem goals are extracted from the originally specified process model using a backward searching approach that is able to identify expected final states from a given BPEL description.

The backward searching feature is particularly important in cases where the final process outcome is a combination of simpler internal outcomes. Starting from the process final states, the algorithm is able to extract the conditions standing at the process end even if the final process outcomes are combinations of outcomes calculated in previous steps. The generated planning problem descriptions are submitted to the Planner Proxy component, which pushes them to the planner. At the end, the generated planning problem solution

*Figure 4. Overview of the internal architecture of the Process Optimizer*



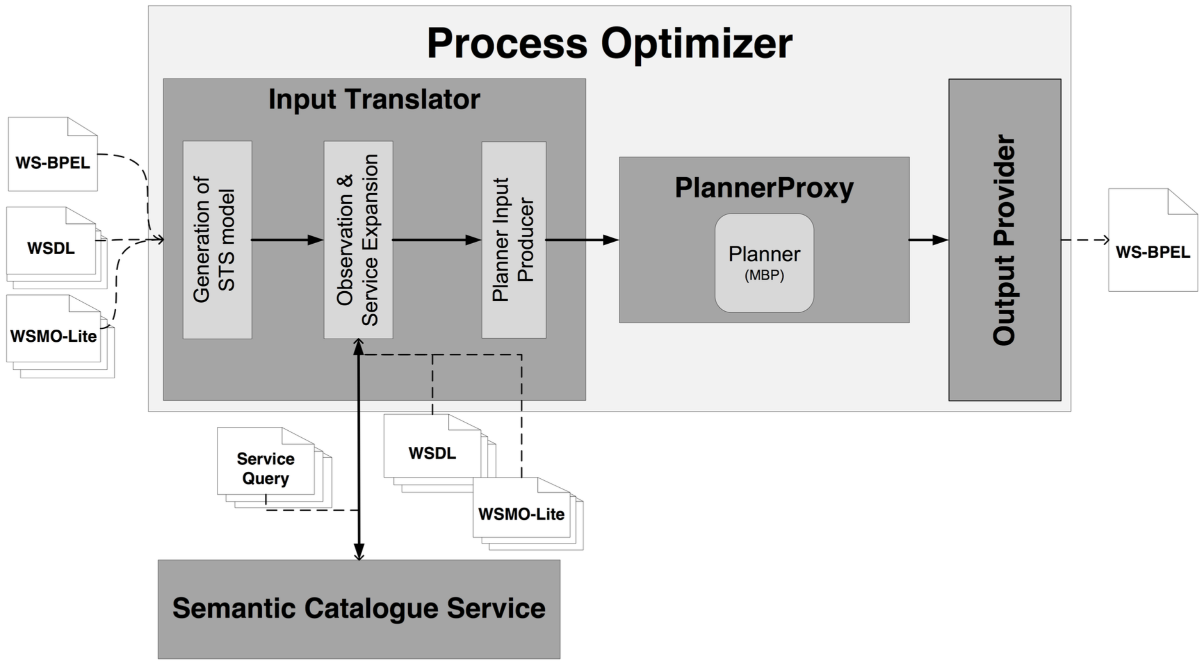
is retrieved from the Outcome Provider, which uses it for extracting the expanded BPEL process.

To better exemplify the expansion process employed by the Process Optimizer, let us consider the BPEL process of Figure 1 represented in detail in Figure 6. Overall, the process consists of activities which are represented in the diagram as rounded boxes, and variables, which are shown as cornered boxes. The control flow of the process is indicated by normal arrows connecting the various activities, while the dashed arrows pointing from the variables to the activities and vice versa display its data flow. The various *assign* activities in the process are used to copy data from one variable to another; the *invoke* activities allow the process to interact with external Web services; the *receive* and *reply* activities are used by the process in order to retrieve the user input and send the final output, respectively. Finally, the structured *sequence* and *flow* activities dictate the order in which their included activities will be executed.

The expansion of the landslide process starts with the transformation into the corresponding STS representation. A simplified illustration of the generated STS model for the nominal process flow (i.e., without any consideration of potential exceptions) is presented in Figure 7. A set of original observations are identified and associated to the end states of service invocations; these include invocations of external services that are not part of a loop construct in the process model.

Each of the identified observations is linked to a specific ontology concept. For example, the observation used for monitoring the outcome of the SOS service returning the precipitation of a given area (activity *Invoke2* in Figure 6), is linked to the geoevents:#precipitation ontology concept. Given an expansion ratio of 0.80 value, the *Observation and Service Expansion Engine* proposes a set of candidate observations, which consists of {geoevents:#precipitation, geoevents:#flow}. This set of candidate observa-

*Figure 5. Process Optimizer execution flow*



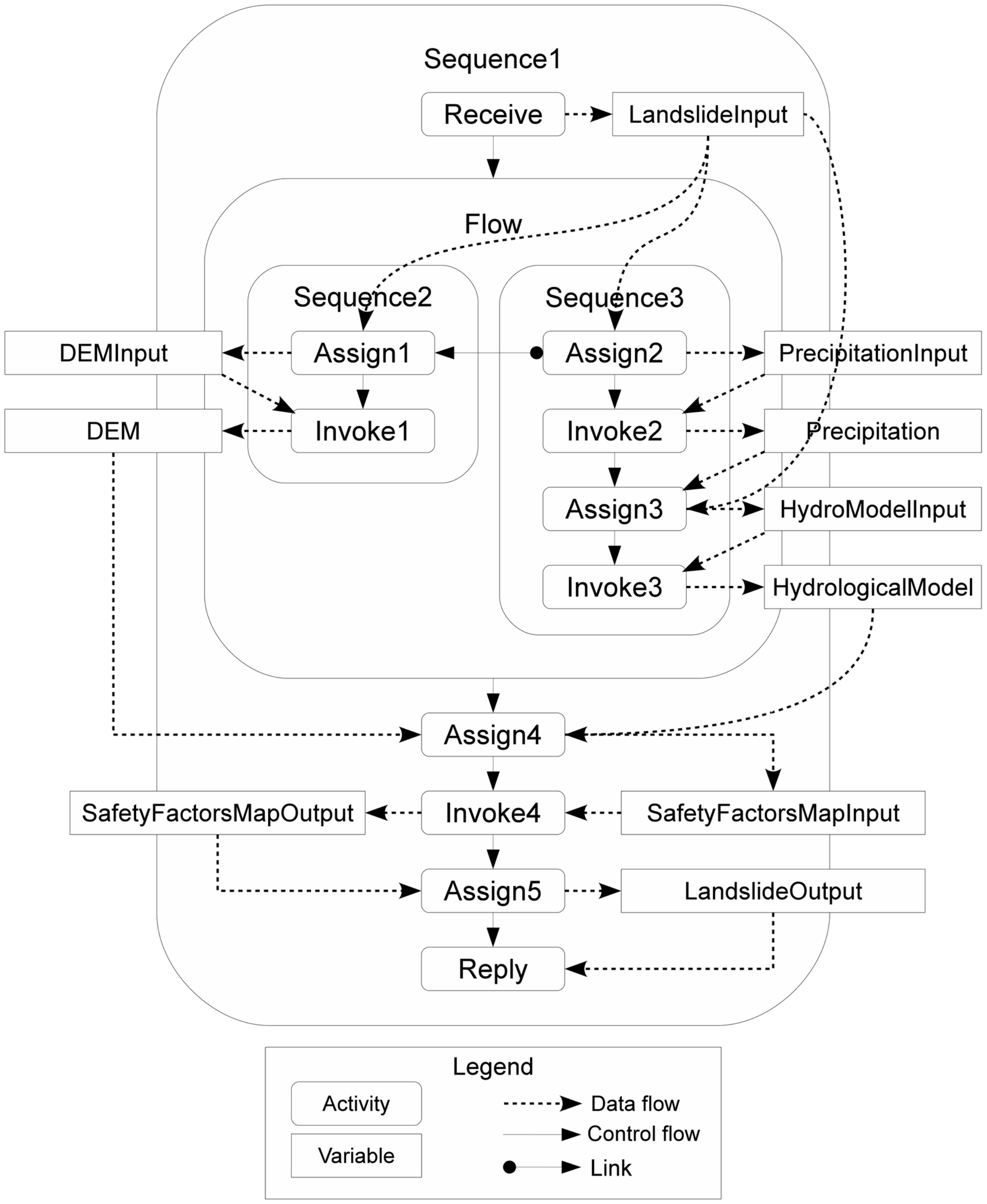
tion concepts provides the input required for the discovery of alternate service chains. Starting from the geoevents:#flow concept and using a forward search strategy, the *Observation and Service Expansion Engine* identifies a candidate service chain that returns the hydrological model. This chain comprises a single WPS service named "Water Flow Level Estimation" that accepts water flow measurements and calculates an estimate of the hydrological model.

Assuming that the input required by this service chain, i.e., the properties monitored by the proposed candidate observation, are available to the landslide process prior to the execution of the SOS precipitation service, then this chain can be used as an optimization to the set of activities specified in *Sequence3*. Along the same lines, similar suggestions can be identified for the rest of the process activities, i.e., in case there are services that can exploit the related candidate observations.

The STS models of the identified candidate service chains and the STS model of the original landslide process are combined in order to formulate the planning problem domain. The extraction of the planning problem goals can be achieved through the discovery of the expected goal states of the landslide process model using the backward search algorithm that has been mentioned before. According to Figure 7, the (nominal) goal state is achieved when the landslide process response is assembled out of the SafetyFactorsMapOutput, which is returned from *Invoke4* activity. Both the domain and goal planning problem descriptions are sent to the planner for the calculation of the planning problem solution.

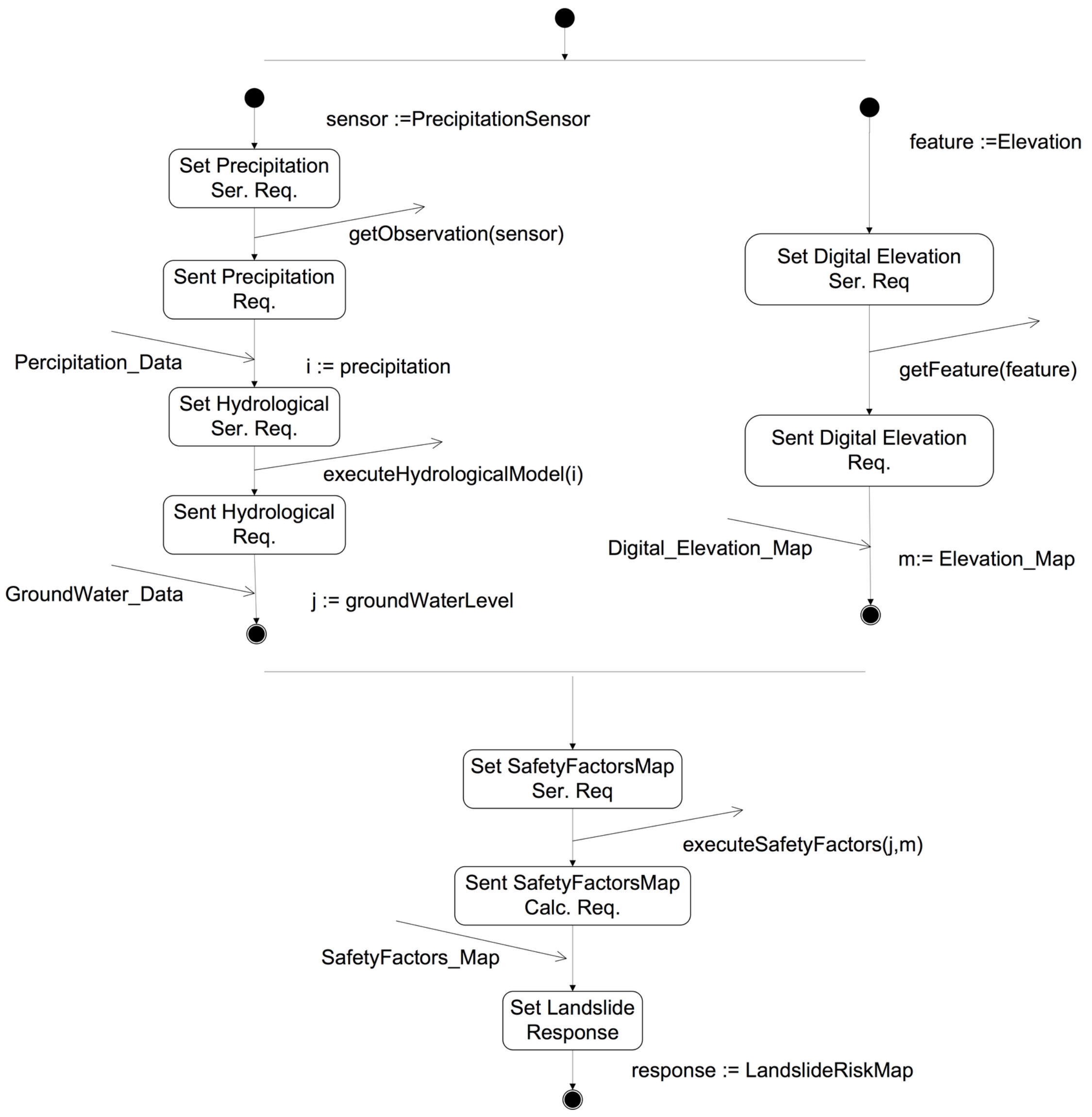
The solution proposed by the planner is generated by the Output Provider and is graphically illustrated in Figure 8. To avoid unnecessary clutter, our example focuses on the extensions introduced to the *Sequence3* activities, but similar extensions can be provided to the whole list of process activities. As it can be seen in Figure 8, the provided extensions (marked with a gray background) include in addition to the original two alternative paths, i.e., these additional paths

*Figure 6. The Landslide BPEL process*



correspond to the activity sequences occurring when floodObs condition is true and the sequence occurring when PrecipitationObs condition is false. These paths enable the exploitation of flow- and/or precipitation-related information, which

*Figure 7. The Landslide Process STS model*


sensor :=PrecipitationSensor
Set Precipitation Ser. Req.
getObservation(sensor)
Sent Precipitation Req.
Percipitation_Data
i := precipitation
Set Hydrological Ser. Req.
executeHydrologicalModel(i)
Sent Hydrological Req.
GroundWater_Data
j := groundWaterLevel
feature :=Elevation
Set Digital Elevation Ser. Req
getFeature(feature)
Sent Digital Elevation Req.
Digital_Elevation_Map
m:= Elevation_Map
Set SafetyFactorsMap Ser. Req
executeSafetyFactors(j,m)
Sent SafetyFactorsMap Calc. Req.
SafetyFactors_Map
Set Landslide Response
response := LandslideRiskMap

may become available on the SCS Engine, for the adaptation of the landslide process.

For example, in the case of flow related information emerging at the SCS Engine prior to the execution of *Invoke2* activity (i.e., the [floodObs==true] condition is valid), the Service Orchestration Engine will invoke *Invoke6* activity. Considering that the alternative path includes a smaller set of activities, it is probable that this leads to smaller execution times. Similarly, the emergence of precipitation-related information to the SCS Engine prior to the execution of the *Invoke2* activity can save execution time, as the invocation of the corresponding SOS will be skipped.

## Semantic Context Space Engine

The Semantic Context Space (SCS) Engine facilitates the provisioning of adaptable processes by offering an open mechanism for the collection and sharing of external information elements. Information elements refer to structured, annotated data, i.e., semantically and/or spatio-temporally, and contained within a specific space. The provided mechanism is independent of the metadata primitives used for the annotation of information elements and supports their logical organization into groups, similarly called scopes. The SCS Engine provides its clients with a basic set of operations, which include writing, grouping, and retrieving of information elements. Specifically the core features of the SCS Engine are:

- The acquisition of semantically and spatio-temporally enhanced information elements. The need for the semantic annotations leads to the support of WSML (Steinmetz & Toma, 2008) and RDFS (Brickley, Guha, & Botts 2004) meta-information models along with associated meta-information search engines.
- The support for the logical grouping of information, the so-called *"information islands"* (e.g., information pertaining to weather conditions), as well as the specification of associations among those information scopes.
- The provision of a loosely coupled coordination model, and more particularly, a subscribe-notify model, which ensures the decoupling between the client and the SCS Engine.

Each information entity stored in the SCS Engine abides by a specific form, which is illustrated in Figure 9. In particular, the main attributes of an information entity are: a unique *identifier Id* of each information element, a *Lease* that represents a fixed period of time in which the information element is considered to be valid, and a set of *MetaInformation objects*, which are responsible for holding the attributed meta-information properties. Instances of the *Scope* class are used for maintaining details about the logical groups that an information entity pertains to.

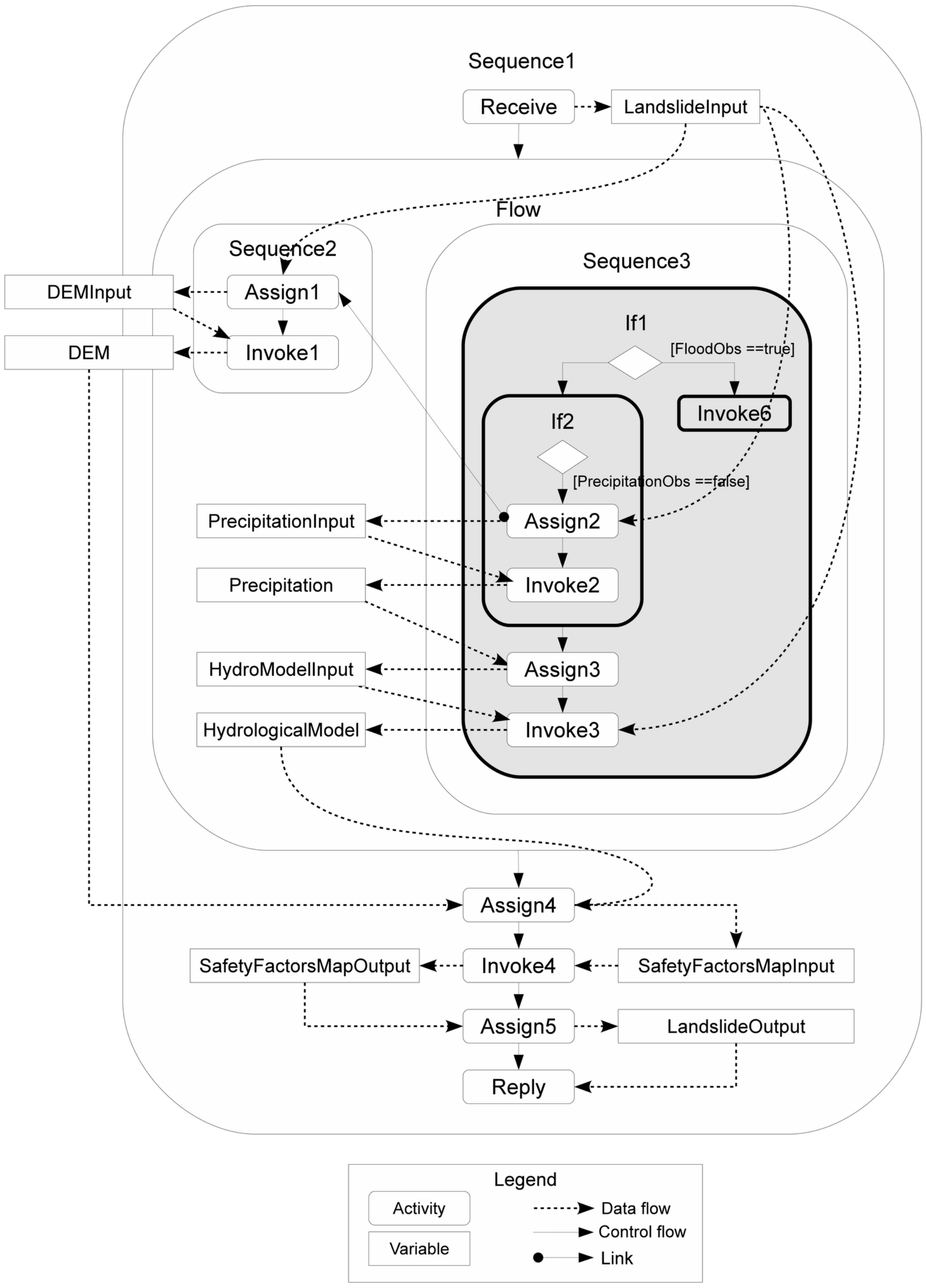
The MetaInformation class is further refined via the *RDFSMetaInformation* and *WSMLMetaInformation* classes that hold semantic extensions described in RDFS and WSML notations respectively, as well as the *SpatialFeature* and *TemporalFeature* classes which store spatial and temporal characteristics about the inserted information accordingly. The implementation can be easily extended so as to offer other types of MetaInformation if needed.

The acquisition mechanism is independent of the metadata primitives used for the annotation of information elements, and supports their logical organization into groups, also referred to as *scopes*. The SCS Engine provides its connected clients, i.e., external data sources, with a basic set of operations, which support the efficient writing, grouping, and retrieving of information elements. The latter can be enhanced with the addition of semantic, spatial, and temporal metadata annotations.

Let us exemplify the role of the SCS Engine in the execution of the expanded Landslide BPEL process of Figure 8. For the sake of our example, we assume that a sensor is available and plays the role of an external source. We also assume that we have available an application that wraps the sensor. This application is directly connected to the RMI interface provided by the SCS Engine and can execute the provided operations. It is also aware of the landslide ontology. To keep the example simple, we decided to omit using spatiotemporal annotations, so we assume that the sensor is located in the same area that the SOS refers to.

Figure 10 graphically illustrates the described example showing SCS Engine's interaction with a sensor playing the role of an external source and the Service Orchestration Engine respectively. The sensor periodically gathers the precipitation value of the area and writes this value along with its meta-information in the SCS. In detail, the sensor

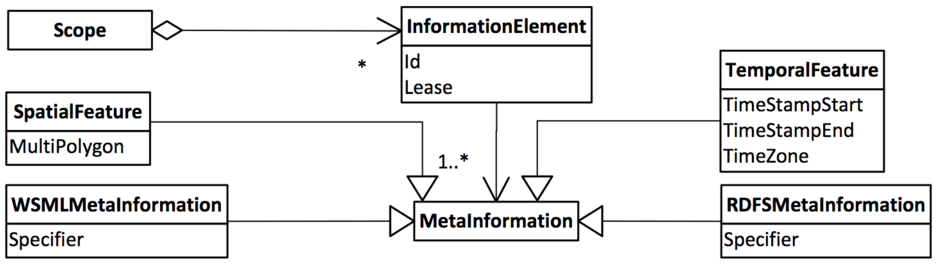
*Figure 8. Expanded version of the Landslide BPEL process*



performs a write operation with input of type: < valueOfPrecipitation, geoevents:#precipitation>.

Taking into account Figure 8, let us suppose that the process starts executing *Sequence3*. In the evaluation of the *If2* condition, the Service

*Figure 9. Information model of the SCS Engine*



Orchestration Engine needs the value of the precipitation. The Service Orchestration Engine will first search in the SCS Engine for the precipitation value. As the Service Orchestration Engine is integrated with the SCS Engine, this is translated into a simple read call in the SCS for values which are annotated with the meta-information geoevents:#precipitation. If the read operation returns a value for the given query, then the *Assign2* and *Invoke2* operations will not be executed, as the Precipitation data is already available through the result of that operation. This way, we save time by omitting the execution of the SOS (*Invoke2*).

## Service Orchestration Engine

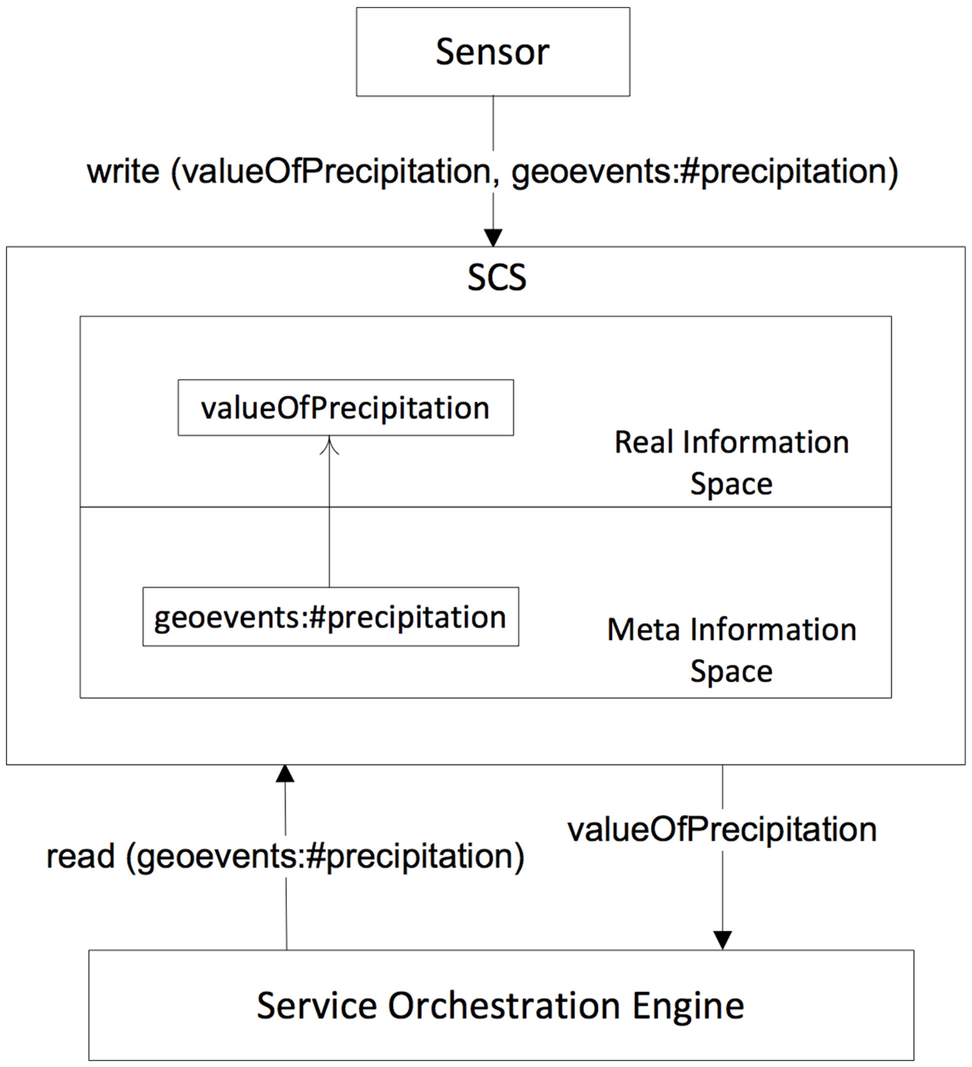
The Service Orchestration Engine is the main component of the ENVISION Adaptive Execution Infrastructure, and is responsible for the decentralized execution of environmental science models that are implemented and deployed as BPEL processes. Central to its architecture is the underlying P2P infrastructure, dubbed *P2P Engine* hereinafter, which implements a binary *hypercube* topology to organize an arbitrary number of available nodes. Each node hosts an instance of the Service Orchestration Engine and cooperates with the rest of the available nodes in the hypercube for the distributed deployment, execution, and monitoring of BPEL processes.

Figure 11 illustrates a complete three-dimensional binary hypercube topology. The number on each edge denotes the dimension in which the two connected nodes are neighbors, while each node is identified by its position, which is conveniently given in Gray code. In general, a complete binary hypercube consists of $N = 2^d$ nodes, where, $d$ is the number of dimensions equaling to the number of neighbors each node has. Hence the network diameter, i.e., the smallest number of hops connecting two most distant nodes in the topology is $D = \log_2 N$.

Hypercubes have been widely used in P2P computing (Schlosser, Sintek, Decker, & Nejdl, 2002; Ren, Wang, & Liu, 2006; Anceaume et al., 2008), and are particularly known for a set of attributes, which are also fundamental for the applicability of our approach:

- **Network Symmetry:** All nodes in a hypercube topology are equivalent. No node incorporates a more prominent position than the others, while any node is inherently allowed to issue a broadcast. Consequently, in our case, any node can become the entry point for the deployment and execution of a process.
- **Efficient Broadcasting:** It is guaranteed that, upon a broadcast, a total of exactly $N - 1$ messages are required to reach all $N$ nodes in the hypercube network, with the last ones being reached after Δ steps, regardless of the broadcasting source. Since broadcasts are extensively used in our approach for the deployment and

*Figure 10. Interaction of the SCS Engine with an external source and the Service Orchestration Engine*

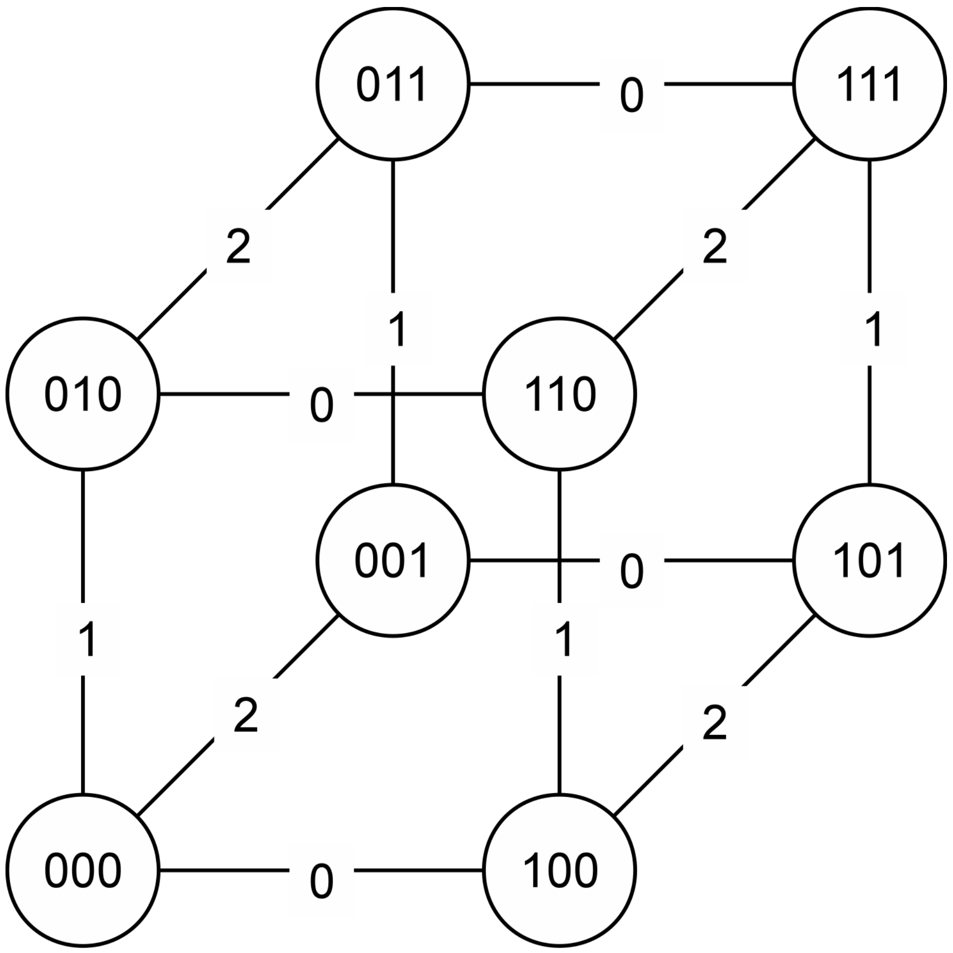


un-deployment of BPEL processes, this property proves to be critical in terms of performance.

- **Cost-Effectiveness:** The topology exhibits $O$ ($\log_2 N$) complexity with respect to the messages that have to be sent, for a node to join or leave the network. Hence, the execution of the respective join and leave protocols does not inflict the overall performance of the distributed BPEL engine.
- **Churn Resilience:** It is always possible for the hypercube topology to recover from sudden node losses. This makes possible the deployment of the distributed BPEL engine in less controlled WAN environments, if needed, where churn rates are naturally higher than the ones met in centrally administered LANs.

Each node participating in the P2P Engine is capable of executing one or more individual BPEL activities as part of one or more process instance executions, while also maintaining one or more of the instance data variables. Thus, one or more

*Figure 11. A three-dimensional hypercube topology*



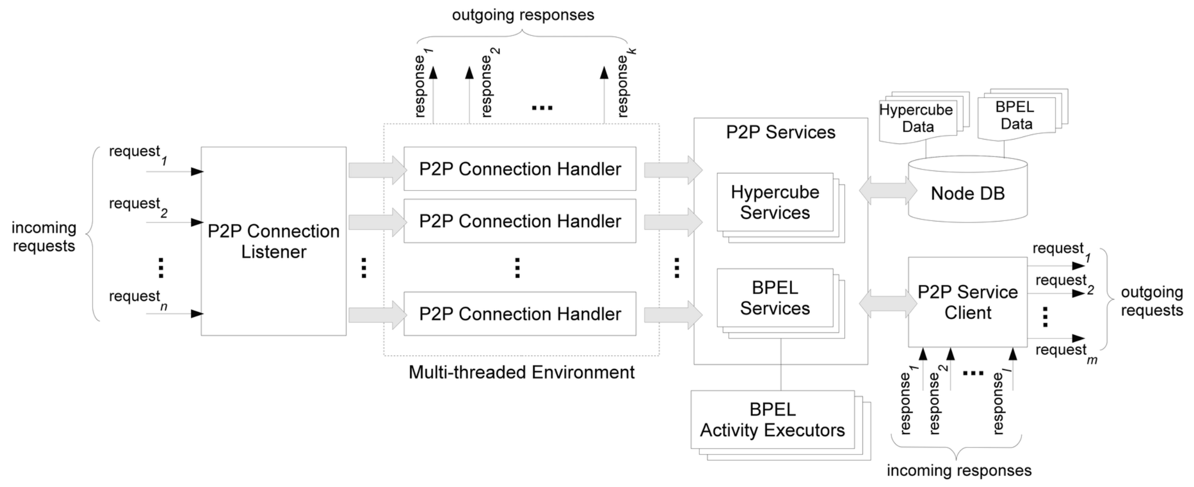
nodes are recruited to contribute in the execution of a given process instance, and coordinate with each other in a completely decentralized manner that is exclusively driven by the structure of the corresponding process.

The main internal components of a node participating in the P2P Engine are shown in Figure 12. The *P2P Connection Listener* acts as the entry point of each node accepting incoming requests from other nodes in the hypercube. Each request is bound to a new P2P connection, which is then passed to a *P2P Connection Handler* for further processing. Since the latter runs in a separate thread, it is possible for a node to simultaneously serve more than one incoming request.

Depending on its type, a request is always associated with a particular *P2P service*, which the P2P Connection Handler selects, instantiates, and executes. P2P services fall into two distinct categories:

1. **Hypercube Services:** Used by the node to perform the various tasks needed for the maintenance of the hypercube topology. Such tasks implement the join and leave algorithms of the hypercube protocol, as well as additional functionality such as broadcasting, random walks, heartbeat, etc., which is essential for the network.

*Figure 12. Internal architecture and main components of the P2P Engine node*



2. **BPEL Services:** Encapsulate all functionality necessary for the distributed deployment, execution, and monitoring of BPEL processes by the nodes of the P2P Engine. Such services provide for the execution of individual BPEL activities (by employing the appropriate *BPEL activity executors*), the read/write of process variables, the response to notifications such as the completion of an activity or the completion of a process, etc.

P2P services may follow a simple one-way communication, or otherwise implement the request-response pattern, in which case the corresponding P2P Connection Handler is used to send back the response message. The execution of most supported P2P services includes the invocation of one or more P2P services on other nodes within the hypercube. This is typical, for instance, in the hypercube service implementing the broadcast scheme, or the BPEL service that is used to execute a particular activity.

To support such situations, each node is equipped with a *P2P Service Client*, which is responsible for establishing a P2P connection with a specified node and consequently submitting the prepared service request. Finally, the majority of the supported P2P services make use of a local database that is embedded within the node. The database holds all information that is needed by a node to participate in the hypercube topology, and also maintains the various tuples, which are generated upon deployment and execution of a BPEL process.

For a BPEL process to be deployed to the Service Orchestration Engine, a request containing a bundle with all necessary files needs to be submitted to one of the available nodes in the hypercube. In particular, this bundle contains the BPEL process specification, the WSDL interface, the WSDL files of all external services, as well as any potentially required XML schemas and/or XSLT transformation files. Upon receipt of the deploy request, the node first performs a syntactic validation of the included files, and then decomposes the process into its constituent activities and variables. The goal is to generate a convenient process representation that will facilitate its decentralized execution. The BPEL decomposition mechanism relies on the use of Program Dependence Graphs (PDGs) for representing the control, data, and synchronization dependencies of the process activities. From a well-formed PDG, it is then easy to decompose the original BPEL process (i.e., to identify the individual process activities and variables) by simply traversing the graph structure.

Before starting out the execution of the process, a distributed recruitment algorithm is carried out (Pantazoglou, Pogkas, & Tsalgatidou, 2013). In a nutshell, the algorithm exploits the hypercube structure to visit the least recently used nodes and appoint them responsible for the constituent activities of the BPEL process. At the same time, the algorithm ensures that the node responsible for the execution of a given activity will be also responsible for holding any output data produced by that activity. In this way, the transfer of voluminous data in between nodes is minimized to the maximum possible extent.

Let us illustrate how the node recruitment algorithm works in the case of the landslide BPEL process of Figure 6. For the sake of simplicity in our example, we assume that the Service Orchestration Engine has just started and comprises a hypercube of eight nodes (3-cube). Figure 13, read from left to right and top to bottom, demonstrates the sequence in which the hypercube nodes are visited upon receipt of an execution request by node 000, while Table 1 shows the recruitment results, i.e., the distribution of the BPEL activities and variables to the hypercube nodes. As it can be seen, the recruitment algorithm managed to engage all available nodes while taking into account their frequency of use upon distribution of the workload.

Let us now describe how the structured activity *Sequence2* of the landslide BPEL process will be executed based on the results of the recruitment procedure shown in Table 1. In principle, a *sequence* activity within a BPEL process is responsible for sequentially executing all its child activities. In our example, node 010, which is responsible for the execution of *Sequence2*, sends an *ExecuteActivity* request to node 011, and waits until it receives back an *ActivityCompleted* notification. Node 011 is responsible for the execution of activity *Assign1*, which is the first child activity of *Sequence2*.

Since the activity *Assign1* is synchronized with activity *Assign2* through a BPEL link (see arrow in Figure 6), node 011 will wait until an *ActivityCompleted* notification is sent from node 001. Then, it sends a *ReadVariable* request to node 000, in order to retrieve the value of the *LandslideInput* variable. After that, the node proceeds with the execution of the copy statements within the *assign* activity, and locally writes the produced outcome to the *DEMInput* variable. At this point, the *Assign1* activity has completed and node 011 sends an *ActivityCompleted* notification to node 010, which is in charge of the parent activity, *Sequence2*.

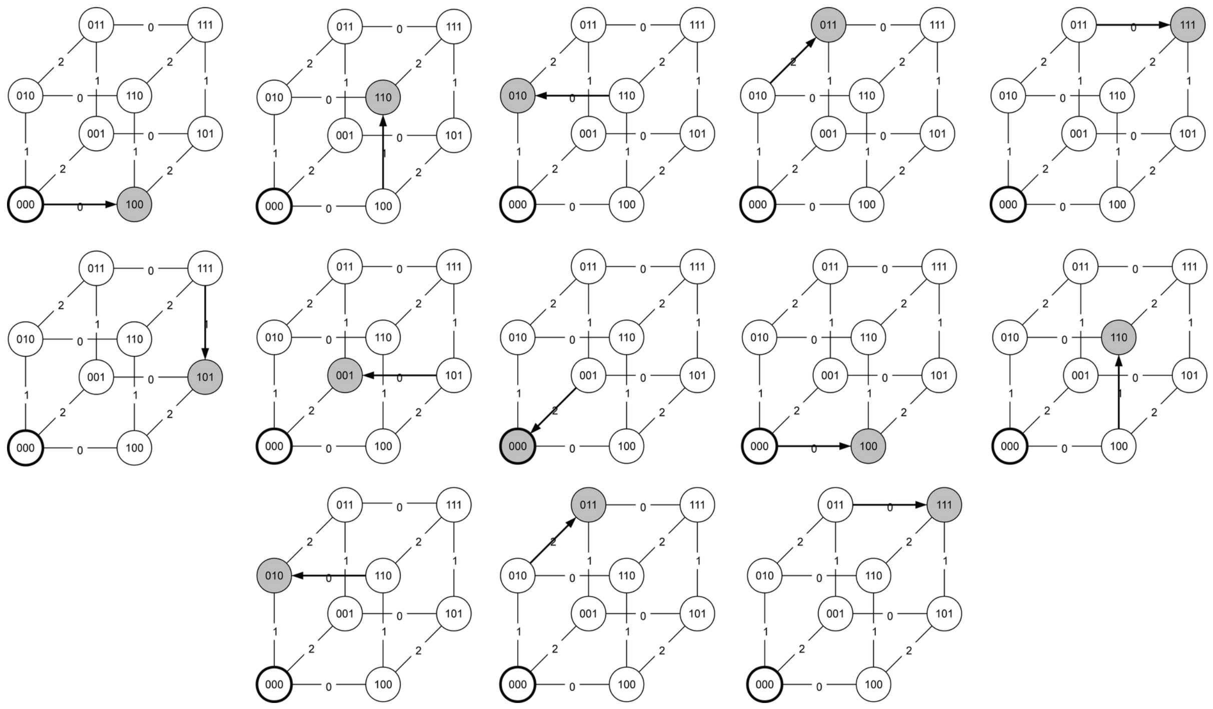
Node 010 resumes the execution of *Sequence2*, which dictates that the *Invoke1* activity is executed next. To do so, an *ExecuteActivity* request is sent to node 111, which is responsible for that activity. Before performing the actual invocation of the Digital Elevation Model WCS, node 111 retrieves the required input by reading the *DEMInput* variable from node 011. After invocation, the service output is locally written to variable *DEM*, and an *ActivityCompleted* notification is sent back to node 010, allowing it to complete the execution of activity *Sequence2*, and send the appropriate notification to node 110, which is in charge of the execution of the parent *Flow* activity.

A number of experiments detailed in a previous publication (Pantazoglou, Pogkas, & Tsalgatidou, 2013) attest that the described distributed way of BPEL process execution outperforms centralized and clustered approaches in the case of BPEL processes that are long-running, involve the exchange of voluminous data with external Web services, and are concurrently accessed by large numbers of users.

## FUTURE RESEARCH DIRECTIONS

The results of this work could be extended and/or improved in numerous ways. The implementation of a WPS interface on top of the Adaptive Execu-

*Figure 13. Recruitment of workers for the execution of the landslide BPEL process*



tion Infrastructure will render it more aligned to the ongoing developments in the environmental science domain. Support for data provenance, as well as the provisioning of efficient big-data transferring mechanisms would further enhance our results and help transforming the Adaptive Execution Infrastructure from a functional prototype into a full-fledged solution for the environmental science domain and beyond.

Finally, we are interested in extending the hypercube-based architecture to support Cloud-based deployment of the Service Orchestration Engine. We anticipate that by moving the Adaptive Execution Infrastructure to the Cloud, we will be able to exploit elasticity capabilities for dynamically increasing or decreasing the hypercube dimension. This way, the execution engine will

*Table 1. Recruitment procedure results*

| Hypercube Node | Assigned Activities | Assigned Variables |
|---|---|---|
| 000 | Receive, Reply, Invoke2 | LandslideInput, Precipitation |
| 100 | Sequence1, Assign3 | HydroModelInput |
| 110 | Flow, Invoke3 | HydrologicalModel |
| 010 | Sequence2, Assign4 | SafetyFactorsMapInput |
| 011 | Assign1, Invoke4 | DEMInput, SafetyFactorsMapOutput |
| 111 | Invoke1, Assign5 | DEM, LandslideOutput |
| 101 | Sequence3 | - |
| 001 | Assign2 | PrecipitationInput |

be able to more effectively respond to workload changes in a timely manner.

## CONCLUSION

The presented Adaptive Execution Infrastructure revolves around the implementation of the following two innovative features:

- **Data-Driven Adaptation:** Running instances of the deployed BPEL processes are aware of information, which may come from external sources (i.e., third-party entities that were not anticipated at design-time). Moreover, they can leverage such data in order to alter their execution, and thereby enhance the performance of the execution infrastructure. Typical scenarios of data-driven adaptation include the cross-instance data re-use and sharing, the invocation of alternative services, the skipping of time-consuming activities, etc. Our Adaptive Execution Infrastructure is equipped with the mechanisms necessary to support adaptation by exploiting semantic and spatiotemporal annotations on the available data.
- **Decentralized Execution:** A distributed architecture based on the hypercube P2P topology along with a set of algorithms that enable the decentralized execution of BPEL processes has been implemented. Our approach targets the improvement of the average process execution times and the enhancement of the overall throughput of the execution infrastructure, in the presence of multiple long-running process instances that involve the exchange of large data. Such cases are typical in many applications, as well as in the environmental science domain, where we validated our approach. The presented algorithms support the decomposition of a given BPEL process and the subsequent assignment of the constituent activities and data variables to the available hypercube nodes. Execution is then performed in a completely decentralized manner without the existence of a central coordinator.

From our experience with the BPEL processes that were developed in the context of the ENVISION pilots, those features are deemed important to ensure a smooth, scalable and efficient execution environment without requiring the involvement of the end-user. The qualitative evaluation of our Service Orchestration Engine on the basis of the landslide ENVISION pilot demonstrated the benefits accruing from our approach to decentralized execution with the use of hypercubes.

Besides, by applying the data-driven adaptation to the same pilot, we verified that such approach holds a potential for future use by many applications in the environmental science domain, as it contributes to the improvement of execution times. Moreover, by leveraging external information and data that can be easily inserted to the system, this feature allows for smart alteration of a model at runtime, without any additional workload imposed on the model's designer or end-user. Indeed, as it was found out through the comparison of our overall approach to a number of renowned solutions for scientific workflow enactment, the Adaptive Execution Infrastructure is currently the only solution encompassing all the aforementioned features, in response to the requirements set by the project.

## REFERENCES

Ai, L., Tang, M., & Fidge, C. (2011). Partitioning composite web services for decentralized execution using a genetic algorithm. *Future Generation Computer Systems*, *27*(2), 157–172. doi:10.1016/j.future.2010.08.003

AllcockW.BresnahanJ.KettimuthuR.LinkM. DumitrescuC.RaicuI.FosterI. (2005). The Globus striped GridFTP framework and server. In Proceedings of the ACM/IEEE Conference on Supercomputing. Washington, DC: ACM/IEEE. 10.1109/SC.2005.72

AltintasI.BarneyO.Jaeger-FrankE. (2006). Provenance collection support in the Kepler scientific workflow system. In Proceedings of the 2006 International Conference on Provenance and Annotation of Data. Berlin: Springer-Verlag. 10.1007/11890850_14

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., & Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of SS-DBM*, (pp. 423–424). IEEE Computer Society. doi:10.1109/SSDM.2004.131124110.1109/SSDM.2004.1311241

Anceaume, E., Ludinard, R., Ravoaja, A., & Brasileiro, F. V. (2008). Peercube: A hypercube-based p2p overlay robust against collusion and churn. In S. A. Brueckner, P. Robertson, & U. Bellur (Eds.), *Proceedings of the Second IEEE International Conference on Self-Adaptive and Self- Organizing Systems* (pp. 15-24). IEEE Computer Society.

AthanasopoulosG.TsalgatidouA. (2010). An Approach to Data-Driven Adaptable Service Processes. In Proceedings of the 5th International Conference on Software and Data Technologies. Athens, Greece: Academic Press.

Baresi, L., Maurino, A., & Modafferi, S. (2006). Towards distributed BPEL orchestrations. *Electronic Communications of the EASST, 3*.

BertoliP.CimattiA.Dal LagoU.PistoreM. (2003). Extending PDDL to non-determinism, limited sensing and iterative conditional plans. In Proceedings of ICAPS Workshop on PDDL. Trento, Italy: ICAPS.

Brickley, D., Guha, R. V., & Botts, M. (2004). *RDF Vocabulary Description Language 1.0 RDF Schema, W3C Recommendation*. Retrieved from http://www.w3.org/TR/rdf-schema/

Callaghan, S., Deelman, E., Gunter, D., Juve, G., Maechling, P., & Brooks, C. et al. (2010). Scaling up workflow-based applications. *Journal of Computer and System Sciences*, *76*(6), 428–446. doi:10.1016/j.jcss.2009.11.005

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web Service Description Language (WSDL) 1.1*. World Wide Web Consortium (W3C). Retrieved June 17, 2013, from http://www.w3.org/TR/wsdl

Clark, J. (Ed.). (1999). *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium (W3C). Retrieved June 17, 2013, from http://www.w3.org/tr/xslt

Clark, J., & DeRoso, S. (Eds.). (1999). *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium (W3C). Retrieved June 17, 2013, from http://www.w3.org/tr/xpath

Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., & Kesselman, C. et al. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, *13*(3), 219–237.

Doyle, A., Reed, C., Harrison, J., & Reichardt, M. (2001). *Introduction to OGC Web Services* (White Paper). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=6209&version=1&format=htm

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory & Practice*. San Francisco, CA: Morgan Kaufmann Publishers.

Gong, L. (2001). JXTA: A Network Programming Environment. *IEEE Internet Computing*, *5*(3), 88–95. doi:10.1109/4236.935182

Jimenez-PerisR.Patino MartinezM.Martel-JordanE. (2008). Decentralized web service orchestration: A reflective approach. In Proceedings of the 23rd Annual ACM Symposium on Applied Computing, (pp. 494-498). ACM. 10.1145/1363686.1363808

Li, G., Muthusamy, V., & Jacobsen, H.-A. (2010). A distributed service oriented architecture for business process execution. *ACM Transactions on the Web, 4*(1).

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., & Mcllairth, S. … Sycara, K. (2004). *OWL-S: Semantic Markup for Web Services*. Retrieved from http://www.w3.org/submission/owl-s/

McGuinness, D., & Harmelen, F. (2004). *OWL Web Ontology Language Overview*. Retrieved June 17, 2013, from http://www.w3.org/TR/owl-features/

Missier, P., Soiland-Reyes, S., Owen, S., Tan, W., Nenadic, A., & Dunlop, I. … Goble, C. (2010). Taverna reloaded. Lecture Notes in Computer Science, 6187, 471-481.

NandaM. G.ChandraS.SarkarV. (2004). Decentralizing execution of composite web services. In VlissidesJ.M.SchmidtD.C. (Eds.), Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (pp. 170-187). ACM.

OASIS. (2007). *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. Retrieved June 17, 2013, from http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

OGC. (2007a). *OGC® Sensor Observation Service*. Open Geospatial Consortium. Retrieved June 17, 2013, from http://www.opengeospatial.org/standards/sos

OGC. (2007b). *OGC® Web Processing Service 1.0.0*. Open Geospatial Consortium. Retrieved June 17, 2013, from http://www.opengeospatial.org/standards/wps

OGC. (2012). *OGC® Web Coverage Service 2.0 Primer: Core and Extensions Overview*. Open Geospatial Consortium. Retrieved June 17, 2013, from http://www.opengeospatial.org/standards/wcs

OMG. (2011). *Business Process Model and Notation (BPMN) Version 2.0*. Retrieved November 24, 2013, from http://www.omg.org/spec/BPMN/2.0/

Pantazoglou, M., Pogkas, I., & Tsalgatidou, A. (2013). Decentralized enactment of BPEL processes. *IEEE Transactions on Services Computing*. Retrieved from http://doi.ieeecomputersociety.org/10.1109/TSC.2013.6

Ren, H., Wang, Z., & Liu, Z. (2006). A hyper-cube based p2p information service for data grid. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing,* (pp. 508-513). IEEE Computer Society.

Schlosser, M. T., Sintek, M., Decker, S., & Nejdl, W. (2002). Hypercup - hypercubes, ontologies, and efficient search on peer-to-peer networks. *Lecture Notes in Computer Science*, *2530*, 112–124. doi:10.1007/3-540-45074-2_11

Sperberg-McQueen, C. M., & Thompson, H. (2000). *XML Schema*. World Wide Web Consortium (W3C). Retrieved June 17, 2013, from http://www.w3.org/XML/Schema

Steinmetz, N., & Toma, I. (Eds.). (2008). *Web Service Modeling Language*. Retrieved June 17, 2013, from http://www.wsmo.org/tr/d16/d16.1/v1.0/

Taylor, I. J., Shields, M. S., Wang, I., & Philip, R. (2003). Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium,* (pp. 16–27). IEEE Computer Society.

VitvarT.KopeckýJ.ViskovaJ.FenselD. (2008). WSMO-Lite Annotations for Web Services. In Proceedings of the 5th European Semantic Web Conference. Berlin, Heidelberg: Springer.

WutkeD.MartinD.LeymannF. (2008). Model and infrastructure for decentralized workflow enactment. In Proceedings of the ACM Symposium on Applied Computing. New York, NY: ACM. 10.1145/1363686.1363712

Yan, J., Yang, Y., & Raikundalia, G. K. (2006). SwinDeW-a p2p-based decentralized workflow management system. *IEEE Transactions on Systems, Man, and Cybernetics Part A*, *36*(5), 922–935. doi:10.1109/TSMCA.2005.855789

YildizU.GodartC. (2007). Towards decentralized service orchestrations. In Proceedings of the ACM Symposium on Applied Computing. New York, NY: ACM Press.

YuW.(2007).Peer-to-peerexecutionofbpelprocesses. In EderJ.TomassenS.L.OpdahlA.L.SindreG. (Eds.), Proceedings of CEUR Workshop, CAiSE Forum (vol. 247). CEUR.

## ADDITIONAL READING

AuT.-C.NauD. A. (2006). The incompleteness of planning with volatile external information. In Proceedings of the European Conference on Artificial Intelligence, pp. 839–840.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2001). Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Seattle, WA, pp. 473-478. Morgan Kaufmann.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Services Architecture. *Working Group Note.* Retrieved from http://www.w3.org/TR/ws-arch/

DaremaF. (2004). Dynamic Data Driven Application Systems: A new paradigm for application simulations and measurements. In Proceedings of the 4th International Conference on Computational Science, pp. 662-669. 10.1007/978-3-540-24688-6_86

DEMAC. (n.d.). DEMAC project. Retrieved November 25, 2013, from http://vsisls1.informatik.uni-hamburg.de//projects/demac/

Dey, K. A. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing*, *5*(1), 4–7. doi:10.1007/s007790170019

Freeman, E., Hupfer, S., & Arnold, K. (1999). *JavaSpaces Principles, Patterns, and Practice: Principles, Patterns, and Practice*. Addison-Wesley.

Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, *7*(1), 80–112. doi:10.1145/2363.2433

Jean Paoli, C.M.-M. (2008). Extensible Markup Language (XML) 1.0. W3C.

Jinghai, R., & Xiaomeng, S. (2004). In J. Cardoso, & P. A. Sheth (Eds.), *A Survey of Automated Web Service Composition Methods* (Vol. 3387, pp. 43–54). Lecture Notes in Computer Science Springer.

KaldeliE.LazovikA.AielloM. (2011). Continual Planning with Sensing for Web Service Composition. In Proceedings of the 25th Conference on Artificial Intelligence, San Francisco, CA, pp. 1198-1203.

Kuter, U., Nau, D., Reisner, E., & Goldman, R. (2007). Conditionalization: Adapting forward-chaining planners to partially observable environments. In M. Boddy, M. Fox, & S. Thiébaux (Eds.), *Proceedings of the Workshop on Planning and Execution for Real-World Systems, Principles and Practices for Planning in Execution*. AAAI Press.

Lirong, Q., Zhongzhi, S., & Fen, L. (2006). Context Optimization of AI planning for Services Composition. In *Proceedings of the IEEE International Conference on e-Business Engineering*, Shanghai, China, pp. 610-617. IEEE Computer Society.

Marconi, A., Pistore, M., Sirbu, A., Leymann, F., Eberle, H., & Unger, T. (2009). Lecture Notes in Computer Science: Vol. 5900. *Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation* (pp. 445–454). Springer.

Mendling, J. (n.d.). Business Process Execution Language for Web Service (BPEL). *Vienna University of Economics and Business Administration.*

Nixon, L., Simperl, E., Krummenacher, R., & Martin-recuerda, F. (2008). Tuplespace-based computing for the semantic web: A survey of the state-of-the-art. *The Knowledge Engineering Review*, *23*(2), 181–212. doi:10.1017/S0269888907001221

Papazoglou, M., & Georgakopoulos, D. (2003). Introduction: Service-oriented computing. *Communications of the ACM*, *46*(10), 24–28. doi:10.1145/944217.944233

Pistore, M., Marconi, A., Bertoli, P., & Traverso, P. (2005). Automated Composition of Web Services by Planning at the Knowledge Level. In Proceedings of Automated Composition of Web Services by Planning at the Knowledge Level.

PistoreM.TraversoP.BertoliP.MarconiA. (2005). Automated Synthesis of Composite BPEL4WS Web Services. In Proceedings of the IEEE International Conference on Web Services, Orlando, Florida, pp. 293- 301. 10.1109/ICWS.2005.27

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Upper Saddle River, New Jersey: Prentice Hall.

Tsalgatidou, A., & Pilioura, T. (2002). An Overview of Standards and Related Technology in Web Services. Distributed and Parallel Databases, 12, 135-162. Springer. Zwegers (Eds.), At your service: An overview of results of projects in the field of service engineering of the IST programme (pp. 67-100). MIT Press Series on Information Systems.

Vogels, W. (2003). Web Services Are Not Distributed Objects. *IEEE Internet Computing*, *7*(6), 59–66. doi:10.1109/MIC.2003.1250585

VukovicM.RobinsonP. (2004). Adaptive, planning-based, Web service composition for context awareness. In Proceedings of the International Conference on Pervasive Computing, Vienna, Austria.

Waldo, J., & Team, J. T. (2000). *The Jini TM Specification* (2nd ed.). Addison-Wesley Professional.

Zeng, L., Lei, H., & Chandramouli, B. (2005). In B. Benatallah, F. Casati, & P. Traverso (Eds.), *Semantic Tuplespace* (Vol. 3826, pp. 366–381). Lecture Notes in Computer Science Springer-Verlag.

## KEY TERMS AND DEFINITIONS

**Context Aware Computing:** Context Aware Computing is a contemporary trend, which accommodates the provisioning of software systems that are able to exploit contextual information, e.g., location, state, mood, in order to accommodate user requirements.

**Data-Driven Adaptation:** The ability of a service-oriented process to use the information available within its environment and adapt its execution accordingly.

**Distributed Process Execution:** The execution of process models in a distributed manner that moves beyond clusters and other centralized approaches.

**Environmental Processes:** Process models addressing environmental science problems normally comprising spatially related types of services to retrieve geospatial information.

**Heterogeneous Services:** Heterogeneous services comprise the contemporary instantiations of the service-oriented model, e.g., Web services, WSRF services, Peer-to-Peer services, Open Geospatial Consortium services, etc.

**Semantic Tuplespace:** Refers to an extended version of the Linda model, where information is annotated with appropriate semantics and provided API is enhanced so as to accommodate and exploit the underlying semantics.

**Service-Oriented Process:** Process models implemented in terms of services.

## ENDNOTES

[1] www.envision-project.eu

[2] OGC is an international industry consortium to develop publicly available interface standards that support interoperable solutions to "geo-enable" the Web, wireless and location-based services and mainstream IT.