

A Chemistry-Inspired Workflow Management System for Scientific Applications in Clouds

Héctor Fernández*, Cédric Tedeschi†, Thierry Priol*

*INRIA

†IRISA. University of Rennes 1 / INRIA

{hector.fernandez, cedric.tedeschi, thierry.priol}@inria.fr

Abstract—With the proliferation of Web Services, scientific applications are more and more designed as temporal composition of services, commonly referred to as, *workflows*. To address this paradigm shift, different workflow management systems have been proposed. If their efficiency has been established over centralized reliable systems, it is questionable over highly decentralized failure-prone platforms. Scientific applications started to be deployed over emerging clouds, leading to new issues, like *elasticity*, *i.e.*, the possibility to dynamically refine at runtime the amount of resources dedicated to an application. This raised a new demand for programming models, able to express autonomic self-coordination of services in a dynamic, *elastic* platform.

Chemistry-inspired computing recently regained *momentum* in this context, naturally expressing parallelism, distribution, and autonomic behaviors. While its high expressiveness and adequacy for this context has been established, the chemical model severely suffers from a lack of proof of concepts. In this paper, we concretely show how to leverage such models in this context. We focus on the design, the implementation and the experimental validation of a chemistry-inspired scientific workflow management system.

Index Terms—Scientific workflows, Workflow management system; Nature-inspired models, Chemical programming

I. INTRODUCTION

Until recently, scientific applications were commonly hard-to-maintain unreadable scripts, leading to a poor reusability level and high maintenance costs. With the proliferation of Web services and the increasing adoption of service-oriented computing, whose primary goal is to make a collection of software services accessible through the network, scientists started to develop their applications as compositions of Web services, today commonly referred to as *workflows*. This shift of paradigm recently led to more reutilization, and experiment sharing in the community. The specification and execution of such workflows are managed by workflow management systems, responsible for the coordination of the services involved. Addressing the limitations of initial workflow languages such as the BPEL business standard [1], different systems, for example Taverna [2], Kepler [3], Triana [4], Pegasus [5] or Askalon [6] provide nice features such as implicit parallelism or data-driven coordination, increasing the abstraction regarding execution management while improving the efficiency and manageability of *science* workflows, as formulated by Zhao, Raicu, and Foster in 2008 [7].

With the rise of cloud computing, science workflows started to be deployed over clouds. For example, the Magellan project

is aimed at providing cloud infrastructures for science [8]. Clouds, or not far from here, federation of clouds, are the new target platform for scientific workflows. One key properties of cloud computing is *elasticity*, *i.e.*, the possibility for cloud users, to dynamically adapt the quantity of resources at his disposal at runtime.

Thus it appears that future scientific workflow systems and languages should provide a natural way to express both workflows and platform characteristics. We identify several critical features the future WMS must address: (i) the high degree of parallelism and distribution of services deployed over a federation of clouds, (ii) the potential issues brought by a centralized coordinator such as single point of failure and scalability, and (iii) the elasticity and distribution of clouds, especially when dealing with federations of clouds.

Lately, nature metaphors, and in particular chemistry-inspired analogies have been identified as a promising source of inspiration for developing new approaches for autonomous service coordination [9]. Among them, the *chemical programming paradigm* is a high-level execution model. Within such a model, a computation is basically seen as a set of reactions consuming some molecules of data interacting freely within a *chemical solution* producing new ones (resulting data). Reactions take place in an implicitly parallel, autonomous, and decentralized manner. More recently, the Higher-Order Chemical Language (HOCL) [10] raised the chemical model to the higher-order, providing a highly-expressive paradigm: every entity in the system (in our case data, services and their dependencies, and the platform itself) is seen as a molecule. Moreover, rules can apply on other reaction rules, programs dynamically modifying programs, opening doors to dynamic adaptation. This model is now envisioned as an alternative naturally expressing autonomous coordination [11]. However, while its expressiveness and adequacy to service coordination have been established, the actual experimentation of the chemical model has remained quite limited until now. There is a strong need of a proof of concept to show its viability, in particular compared to current reference WMS.

Contribution. In this paper, we present a workflow management system able to solve a wide variety of workflow patterns both in a centralized and a decentralized way following the chemical model. Its implementation and performance evaluation on different classic scientific workflows presenting different characteristics are discussed. For the sake of comparison

and discussion, workflows tested were also executed on top of Taverna and Kepler WMS, validating our software prototype, and establishing the viability of the concept.

Section II introduces the preliminaries of our work: workflow management systems and the chemical computing paradigm. Section III describes the architecture and workflow engine we have built. We show how a workflow is described in our model, and how chemical rules are defined and combined so that they can solve a wide variety of workflow patterns. Section IV focus on the implementations of both centralized and decentralized versions of the system defined. Section V details the experimental campaign and its results. Section VI discusses related works. Section VII draws a conclusion.

II. BACKGROUND

We now introduce the two main areas of this work: workflow management systems and the chemical computing model.

A. Workflow Management for e-Science

The increase in the reliance on service-oriented architectures (SOA) in e-science resulted in applications to be more and more defined as workflows of services. As a natural consequence, workflow management systems have gained recently considerable attention. The BPEL standard [1] and followers [12] were first briefly adopted by the scientific community. Then, *science-oriented* workflow languages and systems were designed, to cope with the different characteristics of scientific applications, in parallelism high parallelism and need for scheduling. In this way, a number of systems were designed for the expression and execution of scientific workflows. Taverna [2], Kepler [3], Triana [4], Pegasus [5] or Askalon [6] provide nice features such as implicit parallelism or data-driven coordination, increasing the abstraction regarding execution management while improving the efficiency and manageability. All typically provide a visual notation for composition of services.

The remainder of this section introduces the two open-source workflow systems used for the sake of validation of our work: Taverna and Kepler. We chose Kepler and Taverna because they are among the most used and mature open-source scientific systems. As mature as it can be, we choose not to use Pegasus as the resource management is integrated into the workflow manager, which is not of our primary concern here.

Taverna builds upon service-oriented architectures, and the Web service standards. Interactions between services (*processors*) are defined using the XML-based Scuff language or a GUI. Taverna is data-driven, data-dependencies specify links among different services, thus parallelism is implicit. Note that control-dependencies can also be specified by links that define precedence conditions among processors. Taverna's workflow engine is centralized; a unique coordinator manages the coordination of all computation blocks.

Kepler is a centralized workflow engine built upon Ptolemy II [13] by the members of the Science Environment for Ecological Knowledge (SEEK) project and the Scientific Data Management (SDM) project. It also relies on a data-driven

model for simulating, and designing of real-time and concurrent workflows using a proprietary modeling markup language called MoML. This language is based on the actor-oriented modeling paradigm which consists in a composition of computation blocks called *actors* that represent operations or data sources. Due to its data-driven behavior, Kepler provides an intuitive and implicit parallel execution, however it hinders the execution of complex workflow patterns. In recent versions, some control structures can be supported through somewhat sophisticated programming.

The limitation of both Taverna and Kepler is the lack of (i) facilities to describe more complex control-flow patterns, and (ii) support for decentralized coordination of the workflow execution.

B. Chemical Programming

Nature analogies, and more specifically bio-chemical metaphors, have recently gained momentum in the construction of programming models coping with the requirements of the Internet of Services [9]. Initially proposed to naturally express highly parallel programs, the chemical programming paradigm exhibits properties required in emerging service platforms and naturally express autonomic coordination.

According to the chemical metaphor, molecules (data) float in a chemical solution, and react according to reaction rules (program) producing new molecules (resulting data). These reactions take place in an implicitly parallel, autonomous, and non-deterministic way until no more reactions are possible, a state referred to as *inertia*. The computation is carried out according to local conditions without any central coordination, ordering or serialization. This programming style allows writing programs cleared of any artificial sequentiality and to concentrate on the functional aspects of the problem solved. The presence of a molecule is enough to trigger a reaction requiring such a molecule. Nevertheless, as it will be shown, we can express sequentiality if needed.

The GAMMA language [14] first formalized chemical programming. In GAMMA, the solution is a multiset containing the molecules, and reactions between molecules are rules rewriting the multiset. The multiset is the unique data structure. More recently, higher-order chemical programming has been proposed, through HOCL (*Higher Order Chemical Language*) [10]. In HOCL, every entity is a molecule, including reaction rules. A program is a solution of molecules, formally a multiset of atoms, denoted A_1, A_2, \dots, A_n , “,” being the associative and commutative operator of construction of compound molecules. Atoms can be constants (integers, booleans, *etc.*), reaction rules, tuples of n atoms, denoted $A_1:A_2:\dots:A_n$, or sub-solutions, denoted $\langle M_i \rangle$, where M_i is the molecule content of the sub-solution. A reaction involves a reaction rule **replace P by M if V** and a molecule N satisfying the pattern P and the reaction condition V . The reaction consumes the molecule N to produce a new molecule M . This rule can react as long as a molecule satisfying the pattern P exists in the solution. Its *one-shot* variant, denoted **one P by M if V** , reacts only once, and is consumed in

the reaction. Rules can be named or appear explicitly in the solution. Let us consider the simple HOCL program below that extract the maximum even number from a set of integers.

```

1.01 let selectEvens = replace x, ω by ω if x%2 != 0 in
1.02 let getMax = replace x, y by x if x ≥ y
1.03 in
1.04 ⟨
1.05   ⟨selectEvens, 2, 3, 5, 6, 8, 9⟩,
1.06   replace-one ⟨selectEvens = s, ω⟩ by getMax, ω
1.07   ⟩

```

The *selectEvens* rule removes odd numbers from the solution, by repeated reactions with an integer x, ω denoting the whole solution in which *selectEvens* floats, deprived of x . The *getMax* rule reacts with two integers x and y such that $x \geq y$ and replaces them by x . In a solution of integers, this rule, by its repeated application extracts the maximum value. The solution is composed by (i) a sub-solution containing the input integers along with the *selectEvens* rule, and (ii) a higher-order rule (on Line 1.06) that will *open* the sub-solution, extract the remaining (even) numbers and introduce the *getMax* rule.

Solving the problem requires the sequentiality of the reactions of the two rules. This can be achieved by the higher-order: in an HOCL program, a sub-solution can react with other elements only if it has reached the state of inertia. In other terms, the higher-order rule will react with the sub-solutions only when no more reactions are possible within it, *i.e.*, when it contains only even numbers. (Note that the order in which odd numbers are deleted is non-deterministic.) The result is then as follows:

```

⟨
  ⟨selectEvens, 2, 6, 8⟩,
  replace-one ⟨selectEvens = s, ω⟩ by getMax, ω
⟩

```

Then, the higher-order rule reacts with it, extracting remaining numbers, introducing dynamically the *getMax* rule, and in this way triggering the second phase of the program where the maximum value is kept. The resulting solution is: $\langle 2, 6, 8, \text{getMax} \rangle$. *getMax* then reacts with pairs of integers until only 8 remains. Note that, due to the higher-order, putting both rules directly in the solution of integers could entail a wrong behavior as the pipeline between the two rules would be broken, possibly leading to a wrong result, for instance if *getMax* reacts first with the 8 and 9, thus deleting the 8.

While this example is quite simple, it already provides the intuition behind autonomic coordination and online adaptation.

III. CHEMICAL WORKFLOW MANAGEMENT SYSTEM

In this section, we describe chemistry-inspired workflow management systems. First, the coordination mechanisms developed, which build upon higher-order chemistry, are presented. Then, the architecture underlying it, for both centralized or decentralized coordination are described. The concepts presented in this section take their origin in the founding work presented in [15].

A. Workflow Representation

We now consider a workflow of Web Services (WS). The general shape of the chemical representation of a workflow is as follows: the main solution is composed of as many sub-solutions as there are WSEs in the workflow. Each sub-solution represents a WS with its own data and control dependencies with other WSEs. More formally, a WS is a molecule of the form $WS_i : \langle \dots \rangle$ where WS_i refers to the symbolic name given to the service whose connection details and physical position are hidden.

Let us consider a simple workflow example whose chemical representation is illustrated by Figure 1. It is composed of four services S_1, S_2, S_3 and S_4 . In this example, after S_1 completes, S_2 and S_3 can be invoked in parallel. Once S_2 and S_3 have both completed, S_4 can be invoked. $WS1 : \langle \dots \rangle$ to $WS4 : \langle \dots \rangle$ represent WSEs in the solution. The relations between WSEs are expressed through molecules of the form $DEST:WS_i$ with WS_i being the destination WS where some information needs to be transferred. For instance, we can see in the $WS1$ sub-solution that $WS1$ will transfer some information (its outcome) to $WS2$ and $WS3$ (Line 2.02).

```

2.01 ⟨ // Multiset (Solution)
2.02   WS1:⟨CALL:S1, PARAM:⟨in1⟩, DEST:WS2, DEST:WS3⟩, // WS1 Sub-solution
2.03   WS2:⟨DEST:WS4, replace RESULT:WS1:value1
2.04     by CALL:S2, PARAM:⟨(value1)⟩⟩,
2.05   WS3:⟨DEST:WS4, replace RESULT:WS1:value1
2.06     by CALL:S3, PARAM:⟨(value1)⟩⟩,
2.07   WS4:⟨replace RESULT:WS2:value2, RESULT:WS3:value3
2.08     by CALL:S4, PARAM:⟨(value2)⟩⟩
2.09   ⟩

```

Fig. 1. Chemical workflow representation.

Let us have a more precise look on these dependencies. $WS2$ has a *data* dependency: it requires a molecule $RESULT:WS1:value1$ containing the result of S_1 to be invoked (second part of Line 2.04). The two molecules produced by the reaction represent the call to S_2 and their input parameters. They are expressed using a molecule of the form $CALL:Si$, and a molecule $PARAM:⟨in_1, \dots, in_n⟩$, where in_1, \dots, in_n represent the input parameters to call the service Si . In Figure 1, this input parameter corresponds to the result of some previous service S_j . $WS3$ works similarly. $WS4$ performs a particular control pattern known as *synchronization*. It needs to wait until $WS2$ and $WS3$ have completed, in other words, until the molecules $RESULT:WS2:value2$ and $RESULT:WS3:value3$ appear in its sub-solution, to start its own execution. In addition, a data dependency is also expressed in $WS4$: the result of S_2 is required to call S_4 .

To ensure the execution of a chemical workflow, additional *generic* chemical rules (*i.e.*, independent of any specific workflow) must be defined. These rules consume and generate additional molecules to manage transfer of information between services, condition checking, fault detection, and more complex control flows. To express the whole logic of

a workflow, these rules have to be composed relying on the analogy of *molecular composition*. This concept consists in the composition of several molecules, which are composed based on data molecule dependencies, and whose reactions produce new molecules reacting in their turn, and so on, to complete the workflow.

B. Generic Rules for Invocation and Transfer

Common tasks in a workflow of services are service invocation and information transfer between services. We now review three *generic* rules illustrated in Algorithm 1, responsible for these tasks, and that will be commonly encountered in the compositions presented later. The *invokeServ* rule encapsulates the actual invocation of services. When reacting, it invokes the Web Service S_i , by consuming the tuples $CALL:S_i$ representing the invocation itself, and $PARAM:\langle in_1, \dots, in_n \rangle$ representing its input parameters, and generates the molecules containing the results of the invocation in the WS_i sub-solution. The molecule $FLAG_INVOKE$ is a flag whose presence in the solution indicates that the invocation can take place. The *preparePass* rule is used for preparing the messages to transfer the results to their destination services, that will later trigger the execution of the *passInfo* rule.

Algorithm 1 Basic generic rules.

```

3.01 let invokeServ = replace WSi:⟨CALL:Si, PARAM:⟨in1, ..., inn⟩,
3.02     FLAG_INVOKE, ω ⟩,
3.03     by WSi:⟨RESULT:WSi:⟨value⟩, ω ⟩
3.04 let preparePass = replace WSi:⟨RESULT:WSi:⟨value⟩, DEST:WSj, ω ⟩
3.05     by WSi:⟨PASS:WSj:⟨COMPLETED:WSi:⟨value⟩ ⟩ ⟩
3.06 let passInfo = replace WSi:⟨PASS:WSj:⟨ω1 ⟩, ω2 ⟩, WSj:⟨ω3 ⟩
3.07     by WSi:⟨ω2 ⟩, WSj:⟨ω1, ω3 ⟩

```

Rule *passInfo* transfers molecules of information between WSEs. This rule reacts with a molecule $WS_i:\langle PASS:d:\langle \omega_1 \rangle \rangle$ that indicates that some molecules (here denoted ω_1) from WS_i needs to be transfer to d . These molecules, once inside the sub-solution of d will trigger the next step of the execution. Therefore, the molecule ω_1 will be transferred from sub-solution WS_i to sub-solution d , when reacting with *passInfo* rule.

C. Complex Workflow Patterns

With generic rules described until now, the engine can only support data flows. However, more complex control flows are required to be taken into account, in order to solve a broader range of workflow definitions. We now illustrate how HOCL can be leveraged to deal with complex control flows, by detailing a particular pattern known as *Simple Merge*.

As illustrated in Figure 2, a simple merge pattern is similar to a XOR operation. It involves a structure where two or more *source* service flows converge into a single *destination* in an asynchronous way. The destination service must however be launched only once, regardless of the number of incoming. In other words, only the first source service to complete will influence the remainder of the workflow execution.

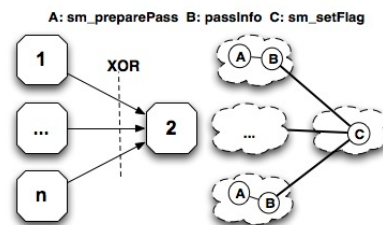


Fig. 2. Simple merge Pattern.

To enhance our workflow engine with the support of the simple merge pattern, we need to define the appropriate generic rules and dispatch them in the sub-solutions of WSEs involved. These rules are given in Algorithm 2. The *sm_preparePass* reaction rule is used to add, in the sub-solution of every incoming service, a particular $MERGE:Yes$ molecule to the information to be transferred to the destination service (see Lines 4.01 and 4.03). The destination WS waits for this molecule and only the first $MERGE:Yes$ molecule received in its sub-solution will be consumed. Next, *sm_setFlag* reaction rule takes place producing one molecule of the form $FLAG_INVOKE$, allowing the service invocation. The following $MERGE:Yes$ molecules received will be ignored. In terms of molecular composition, each source WS will have in its sub-solution one *sm_preparePass* rule (A on Figure 2) and one *passInfo* (B on Figure 2) rules, they are composed with *sm_setFlag* rule (C) in the destination WS.

Algorithm 2 Generic rules - Simple merge pattern

```

4.01 let sm_preparePass = replace DEST:WSj, RESULT:WSi:⟨value⟩
4.02     by PASS:WSj:⟨RESULT:WSi:⟨value⟩, MERGE:Yes ⟩
4.03 let sm_setFlag = replace-one MERGE:Yes by FLAG_INVOKE

```

For space reasons, we omit more complex control flows (such as *synchronization merge*, *exclusive choice* or *discriminator*). However, the support for a wide range of control flow patterns can be found in the research report [16], as well as its design process.

To sum up, the coordination is achieved through a set of autonomic, and local reactions taking place within each WS's sub-solution (or between two WSEs' sub-solutions), providing all the abstractions for a natural expression of a decentralized coordination of virtually all identified workflow patterns [17].

D. Architectural Frameworks

We now show how the chemical engine can be powered over both centralized and decentralized architectures.

1) *Centralized Architecture*: Following the examples of most of Workflow Management Systems mentioned before, the coordination can be managed by a single node, referred to as the *chemical workflow service*, as illustrated by Figure 3. First, notice the S components. They represent the interface with the actual distant WSEs to be called. Then, the workflow definition and coordination information as described before

(i.e., the multiset) is accessed by the chemical engine that will perform the reactions required.

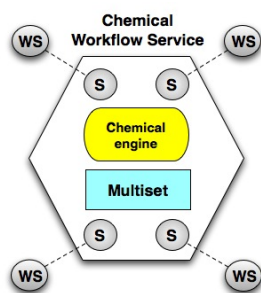


Fig. 3. Centralized chemical WMS architecture.

2) *Decentralized Architecture*: Distribute the control means that each service involved will take its part in the coordination process. Each WS is now chemically encapsulated, to form a *Chemical Web Service (ChWS)*. Each ChWS is equipped with a chemical engine and a local copy of part of the multiset on which its chemical interpreter will act, to realize its part of the coordination. The multiset, containing the workflow definition and thus all required coordination information, will now act as a space shared by all ChWSes involved in the workflow. In other words, ChWSes will communicate by reading and writing it, as illustrated by Figure 4.

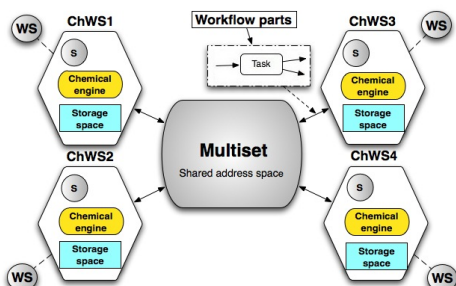


Fig. 4. Decentralized chemical WMS architecture.

IV. SOFTWARE PROTOTYPE

In this section, we discuss the implementation of a software prototype for both previously described architectures. The low layer of our prototype is an HOCL interpreter based on *on-the-fly* compilation of HOCL specifications. The whole prototype is written with Java.

A. Centralized Version

The prototype is illustrated by Figure 5. As mentioned in Section III, the workflow definition is executed as a chemical program by the chemical workflow service. The low layer of the architecture is an HOCL interpreter. Given a workflow specification as input (an HOCL program), it executes the workflow coordination by reading and writing the multiset initially fed with the workflow definition. The interface between the chemical engine and the distant services themselves is realized through the *service caller*. The service caller relies on the DAIOS framework [18], which provides an abstraction

layer allowing dynamic connection to different flavours of services (SOAP or RESTful), at the same time abstracting the target service's internals. DAIOS was specially extended with a module which automatically generates the bindings, as well as input and output messages required between the chemical engine and a Web service.

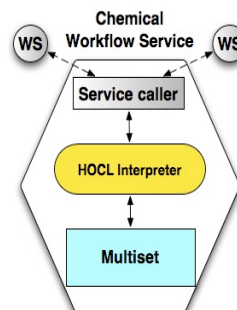


Fig. 5. Centralized architecture.

B. Decentralized Version

The decentralized prototype is illustrated in Figure 6. Basically, the difference between centralized and decentralized implementations is the functionality of the multiset. It now represents a shared space playing the role of a communication mechanism and a storage system. As we detailed before, the workflow definition is comprised of one sub-solution per WS involved. The information in one sub-solution can only be accessed by the ChWS owner of/represented by that sub-solution. On each ChWS, a local storage space acts as a temporary container for the sub-solution to be processed by the local HOCL interpreter. ChWSes communicate with the multiset through RMI interfaces to read and modify it concurrently. Periodically and independently from each other, ChWSes request their sub-solution to this multiset. The sub-solution obtained is then processed, modified by the local HOCL interpreter and then pushed back to the multiset as an update. On the other hand, the *service caller* now represents the interface between a ChWS and a concrete WS.

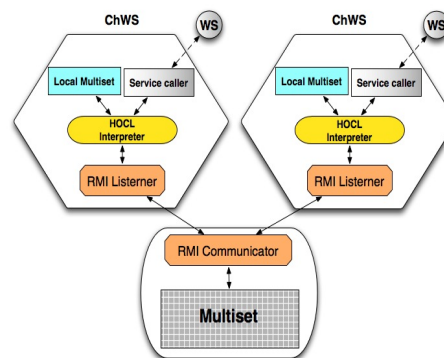


Fig. 6. Decentralized architecture.

To implement the needed communication mechanism, some Java RMI modules are included into the ChWSes and the

multiset. The multiset has been enhanced with an RMI server-communicator module, providing three remote operations for the ChWSes: **put** operation adds molecules in the sub-solution of a specified ChWS, **get** withdraws the content of a ChWS solution, and **isConsumable** determines whether there is any new information to be processed by the calling ChWS. On the ChWS' side, an RMI listener module has been integrated on each ChWS, periodically calling the **isConsumable** operation, which, if it returns **true**, triggers the invocation of the **get** operation to extract the content of the WSi solution which will be copied into its local multiset. Once the HOCL interpreter has finished the execution, internally the RMI listener call the **put** operation to update the content of its sub-solution into the multiset. This multiset implementation presents some similarities with *tuplespaces*, founded in the Linda coordination language [19].

V. PERFORMANCE RESULTS

In this section, we present and analyse our experimental results. Four engines have been used: Taverna Workbench 2.2.0, Kepler 2.0, and both centralized and decentralized version of the HOCL-based workflow engine, referred to as *HOCL Cen.* and *HOCL Dec.* in the following, respectively.

Recall that our objective is not so much comparing performance as is establishing the viability of a chemistry-based workflow engine. In other terms, Taverna and Kepler represent validated standards we use as guidelines.

A. Workflows considered

Three scientific workflows were executed. Illustrated by Figure 7 (left), *BlastReport*³ is a bio-informatics workflow which retrieves a blast report of a protein in a database given its ID. The second one, *CardiacAnalysis*¹, illustrated in Figure 7 (right), is a cardiovascular image analysis workflow which extracts the heart's anatomy from a series of image sequences by applying image processing algorithms, developed by the CREATIS-LRMN biomedical laboratory¹. The third one, *Montage* [20], given in Figure 8 is a classic astronomical image mosaic workflow processing large images of the sky. These workflows present different characteristics such as the number of services involved, the amount of data exchanged and the complexity of the coordination required (data processing included, such as iterations of lists of objects). We attempt to characterize these workflows as follows.

- The *BlastReport* workflow includes 5 services, and presents a medium level of data exchange (simple objects, lists) and low coordination overhead – it is composed mostly of sequences.
- The *CardiacAnalysis* workflow includes 6 services, presenting a high level of data exchange (complex objects, lists) and a high coordination overhead (synchronizations, loop iteration, parallelism). This overhead does not appear on Figure 7 (right), and is due to the re-entrant nature of the services. Indeed, some services produce lists of

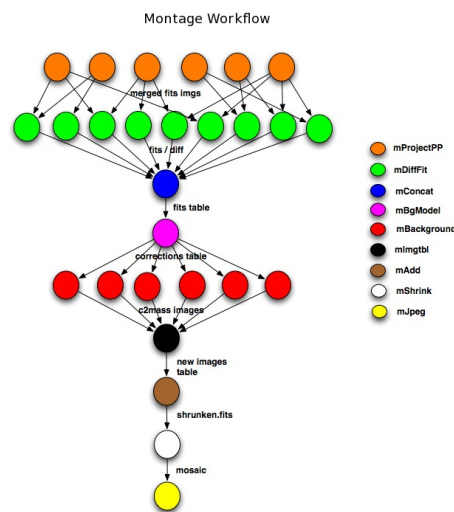


Fig. 8. Montage workflow structure

objects that need to be extracted one by one by iterators, and sent to the next service asynchronously.

- The *Montage* workflow includes 27 services, and exhibits a low rate of data exchange (simple objects) and medium coordination overhead (parallelism and synchronization patterns).

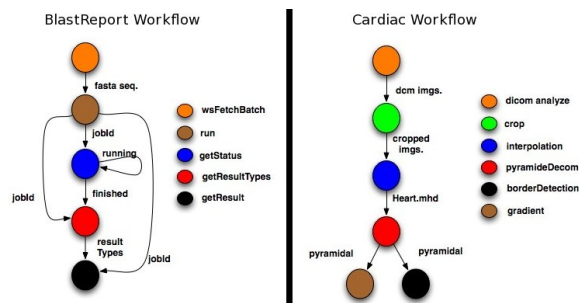


Fig. 7. BlastReport and Cardiac workflows structures

The workflow definitions used for each WMS are available online^{2,3}.

B. Centralized experiments

The workflows were first run using Taverna, Kepler, and HOCL Cen, on a local machine equipped with the Intel core-duo T9600 2.8 Ghz processor and 4GB of memory. Figures 9 and 10 present the results. A first encouraging result is that the execution time for the *Montage* workflow, *i.e.* for a workflow with limited data exchange and coordination overhead, on Kepler, Taverna and HOCL Cen. are quite similar, and even slightly reduced on the HOCL Cen. WMS.

For the *BlastReport* workflow, while results are again similar for the different WMSs, HOCL Cen. takes a little more time. This can be explained by the increased size of the

¹<http://www.creatis.insa-lyon.fr/site/>

²<https://www.irisa.fr/myriads/members/hfernand/hocl/workflows>

³<http://www.myexperiment.org/workflows/2058.html>

multiset representing the *BlastReport* workflow (in terms of number of molecules). However, in terms of ratio, execution times remain very close.

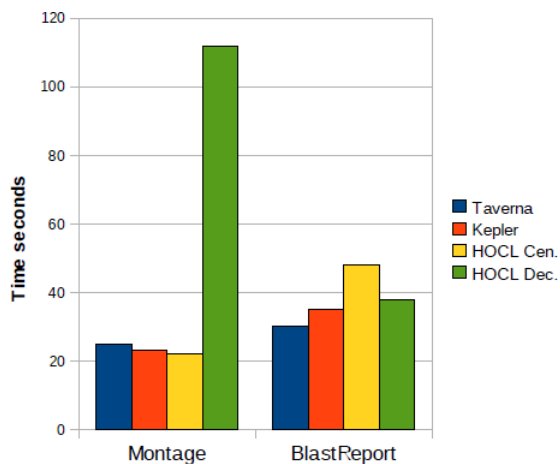


Fig. 9. Performance results, Montage and BlastReport.

Finally, as we can see in Figure 10, the increased coordination overhead of the *CardiacAnalysis*. As mentioned before, this workflow relies on a lot of data processing related to the coordination itself, which, in the case of HOCL Cen., results in a significant increase of the size and processing time of the multiset. Also, no support for parallel execution has been implemented on our current HOCL interpreter version. These two optimization aspects will be investigated in the future.

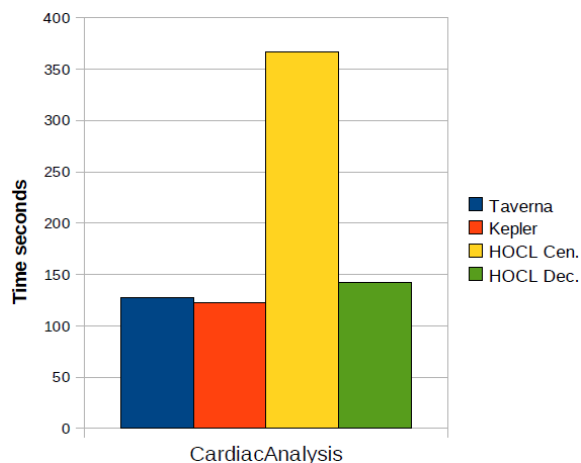


Fig. 10. Performance results, CardiacAnalysis.

C. Decentralized experiments

The workflow were also executed following the HOCL Dec. design. The experiments were conducted on the Grid'5000 platform [21], specifically, on the *paradent* cluster, located in Rennes, each node being equipped with two quad-core Intel Xeon L5148 LV processors, 30 GB of RAM and 40GB InfiniBand Ethernet cards. We now focus on the right-most bar of each series of Figures 9 and 10.

A first observation is the performance degradation using HOCL Dec. on the *Montage* workflow. While the coordination

is executed locally on each ChWS (here the coordination is shared among 27 services), the multiset remains a space shared by every ChWSes leading to potential scalability issues. Also, RMI is known to have different some scalability weaknesses related to memory management [22], in particular provoked by the distributed garbage collector.

On the *BlastReport*, a performance gain over HOCL Cen. is obtained with HOCL Dec., thanks to the distribution of the coordination over the 5 services involved. The number of services participating is not high enough to provoke the previous scalability problems on the multiset side.

For the *CardiacAnalysis* workflow, a considerable performance gain is also obtained using HOCL Dec., demonstrating the benefits of a decentralized workflow execution when workflows present a high coordination overhead like *CardiacAnalysis*. Exploiting the processing resources of each ChWS, the list handling and adaptation tasks are separately managed by each ChWS.

D. Discussion

This series of experiments leads to several conclusions. They constitute a proof of the viability of a chemistry-based engine, as for some workflows, its performance are similar to those of Kepler and Taverna.

Nevertheless, scalability issues need to be tackled. The first limitation comes from the design of the decentralized architecture itself, in which the multiset can constitute a bottleneck. To deal with the decentralization of the multiset itself, we recently formulated solutions based on peer-to-peer protocols, able to distribute and retrieve objects (here, molecules) at large scale [23]. One of the next steps of this work is to build the HOCL Dec. environment on top of such approaches to remove the bottleneck problem, and propose a fully decentralized workflow engine. A more short term possibility is to optimize the communication scheme used, currently based on RMI, also known for some scalability limitations. We will consider relying on a different communication tool.

Recall, beyond performance or optimization considerations, that the chemical models provide all the needed abstractions to naturally express both data-driven and complex control-driven execution, including particular features like cancellation. Please refer to [16] for details. We consider the chemical abstraction as participating in the long term objective of improving the workflow execution models on emerging platform, like clouds, where the elasticity brings new modeling challenges.

VI. RELATED WORKS

Workflow management systems generated a high amount of literature and systems. With the proliferation of service-oriented architectures, and the adoption of the Web service standard, several service-based WMS for scientific applications was proposed, such as Taverna [2], Kepler [3], Triana [4], Pegasus [5] or Askalon [6]. A common limitation of these systems is their lack of support for dynamic adaptation and decentralization, leading to scalability, as well as reliability

issues. Our decentralized architecture share some similarities with the Linda [19] approach, which pioneered the coordination through a shared space. In the same vein, recent works address the need for decentralization in workflow execution. More recently, several approaches replace a centralized BPEL engine by a set of distributed, loosely coupled and cooperating nodes. It has been identified that one promising solution for this is to use a shared tuplespace for coordination [24], [25], [26]. However, in [24], [25], the tuplespace is only used to store data information, our coordination mechanism stores both control and data information into the multiset, which is made possible by the use of the chemical execution model for the coordination of all data and control dependencies. In the scientific area, in [26], authors transform a centralized BPEL definition into a set of coordinated processes. Through a shared tuplespace working as a communication infrastructure, the control and data dependencies exchange among processes to make the different nodes interact between them. Again, the use of BPEL hinders from expressing dynamic and self-adaptive behaviors.

As a more general comment, to our knowledge, these work does not provide real software prototype to experimentally validate their proposal.

VII. CONCLUSION

Scientific applications are now built as workflows of services. Workflow management systems gained recently a lot of attention in this context. However, the emergence of new platforms, such as clouds where elasticity and dynamic adaptation are strong requirements, led to a high demand for new models able to represent both workflows and the platforms, as well as their inherent characteristics.

The chemical model is a promising paradigm naturally capturing parallelism, distribution and dynamics. While the advantages of such a model are now well-established, they still suffer from a lack of proof of concepts and actual deployments.

In this paper, we have proposed a chemistry-inspired workflow management system. A workflow description language and its execution model inspired by such abstractions is discussed. The wide expressiveness (data-flows, control-flows, natural decentralization) of the paradigm is highlighted. Then, its implementation based on the HOCL language, for both centralized and decentralized environment is given. Finally, experiments conducted shown the viability of the concept, lifting a barrier on the path to its actual adoption.

REFERENCES

- [1] OASIS, "Web services business process execution language, (WS-BPEL), Version 2.0," 2007.
- [2] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1067–1100, August 2006.
- [3] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1039–1065, August 2006.
- [4] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The Triana Workflow Environment: Architecture and Applications," in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Secaucus, NJ, USA: Springer, New York, 2007, pp. 320–339.
- [5] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, 2005.
- [6] T. Fahringer, A. Jugravu, S. Pillana, R. Prodan, C. Seragiotto, Jr., and H.-L. Truong, "Askalon: a tool set for cluster and grid computing: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, pp. 143–169, February 2005.
- [7] Y. Zhao and I. Foster, "Scientific workflow systems for 21st century, new bottle or new wine," *IEEE Workshop on Scientific Workflows*, 2008.
- [8] "The magellan research project." <http://magellan.alcf.anl.gov/>, June 2011.
- [9] M. Viroli and F. Zambonelli, "A biochemical approach to adaptive service ecosystems," *Information Sciences*, pp. 1–17, 2009.
- [10] J. Banâtre, P. Fradet, and Y. Radenac, "Generalised multisets for chemical programming," *Mathematical Structures in Computer Science*, vol. 16, no. 4, pp. 557–580, 2006.
- [11] J.-P. Banâtre, T. Priol, and Y. Radenac, "Chemical Programming of Future Service-oriented Architectures," *Journal of Software*, vol. 4, no. 7, pp. 738–746, 2009.
- [12] A. Barker and J. van Hemert, "Scientific Workflow: A Survey and Research Directions," in *PPAM*, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds., vol. 4967. Springer, 2007, pp. 746–753.
- [13] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogenous systems," *Int. Journal in Computer Simulation*, vol. 4, no. 2, pp. 155–182, 1994.
- [14] J.-P. Banâtre and D. L. Métayer, "The gamma model and its discipline of programming," *Sci. Comput. Program.*, vol. 15, no. 1, pp. 55–77, 1990.
- [15] H. Fernández, T. Priol, and C. Tedeschi, "Decentralized Approach for Execution of Composite Web Services using the Chemical Paradigm," in *8th International Conference on Web Services (ICWS 2010)*. Miami, USA: IEEE, July 5-10 2010, pp. 139–146.
- [16] H. Fernández, C. Tedeschi, and T. Priol, "Self-coordination of Workflow Execution Through Molecular Composition," INRIA, Research Report RR-7610, 05 2011. [Online]. Available: <http://hal.inria.fr/inria-00590357/PDF/RR-7610.pdf>
- [17] "The workflow patterns website." <http://www.workflowpatterns.com/>, June 2011.
- [18] P. Leitner, F. Rosenberg, and S. Dustdar, "Daio: Efficient dynamic web service invocation," *IEEE Internet Computing*, vol. 13, no. 3, pp. 72–80, 2009.
- [19] D. G. Y. University, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1.*, pp. 80–112, 1985.
- [20] G. Berriman, E. Deelman, J. Good, J. Jacob, D. Katz, C. Kesselman, A. Laity, T. Prince, G. Singh, and M. hu Su, "Montage: A grid enabled engine for delivering custom science-grade mosaics on demand," in *Proceedings of SPIE Conference 5487: Astronomical Telescopes*, 2004.
- [21] "Grid'5000," <http://www.grid5000.fr>, June 2011.
- [22] M. Alt and S. Gorlatch, "Adapting java RMI for grid computing," *Future Generation Computer Systems*, vol. 21, no. 5, pp. 699–707, May 2005.
- [23] M. Obrovac and C. Tedeschi, "On Distributing the Runtime of the Chemical Programming Model," INRIA, Research Report RR-7661, Jun. 2011. [Online]. Available: <http://hal.inria.fr/inria-00590357/PDF/RR-7610.pdf>
- [24] P. A. Buhler and J. M. Vidal, "Enacting BPEL4WS specified workflows with multiagent systems," in *Proceedings of the Workshop on Web Services and Agent-Based Engineering*, 2004.
- [25] D. Martin, D. Wutke, and F. Leymann, "A novel approach to decentralized workflow enactment," in *Enterprise Distributed Object Computing Conference, IEEE International*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 127–136.
- [26] M. Sonntag, K. Gorlach, D. Karastoyanova, F. Leymann, and M. Reiter, "Process space-based scientific workflow enactment," *International Journal of Business Process Integration and Management*, vol. 5, no. 1, pp. 32 – 44, 2010.