Decentralized Approach for Execution of Composite Web Services using the Chemical Paradigm

Héctor Fernandez, Thierry Priol, Cédric Tedeschi

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Decentralized Approach for Execution of Composite Web Services using the Chemical Paradigm

Héctor Fernández — Thierry Priol — Cédric Tedeschi

N° 7267

Mai 2010

Domaine 3

ISSN 0249-6399 ISRN INRIA/RR--7267--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Decentralized Approach for Execution of Composite Web Services using the Chemical Paradigm

Héctor Fernández , Thierry Priol , Cédric Tedeschi

Domaine : Réseaux, systèmes et services, calcul distribué
Équipes-Projets Myriads

Rapport de recherche n° 7267 — Mai 2010 — 16 pages

Abstract: Nowadays, executions of composite Web services are typically conducted by heavyweight centralized workflow engines. This represents a potential processing and communication bottleneck as well as a single point of failure. In addition to this, centralization induces higher deployment costs, such as a computing infrastructure to support the workflow engine, which is not affordable for a large number of small businesses and end-users. Last but not least, a central workflow engine might expose some security risks such as the business process discovery. More decentralized and dynamic interaction schemes are thus required.

In this paper, we propose a decentralized alternative system for the execution of composite Web services based on an unconventional programming paradigm that relies on the chemical metaphor. It provides a high-level execution model that allows executing composite services in a fully decentralized manner. Our architecture is composed of nodes communicating through a persistent and fault-tolerant shared space containing both control and data flows, allowing to distribute the composition among nodes without the need for any centralized coordination.

Key-words: Service-oriented architectures, service composition, chemical programming

This work was supported by the French National Agency for Research project AUTOCHEM (ANR-07-BLAN-0323) and by the S-Cube European Network of Excellence.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Exécution décentralisée de Services Web composite par le paradigme chimique

Résumé : Au sein des architectures de services actuelles, la coordination des services s'appuient fortement sur des moteurs centralisés. Cette centralisation entraîne des faiblesses de passage à l'échelle et de tolérance aux pannes, et pose des problèmes de sécurité. Plus généralement, le coût de mise en place ou l'accès à une telle infrastructure peut s'avérer trop grand pour de nombreux utilisateurs finaux ou petites entreprises. La décentralisation de cette coordination s'avère donc une piste de recherche à poursuivre.

Dans ce rapport, nous proposons une alternative décentralisée pour l'exécution de Web Services composites, en s'appuyant sur le paradigme de programmation chimique. L'approche que nous proposons fournit un modèle d'exécution haut niveau et totalement décentralisé. L'architecture mettant en place ce modèle consiste en l'utilisation d'un espace partagé contenant l'information sur les flots de contrôle et de données liées à la coordination, et au travers duquel les services mis en jeu vont communiquer.

Mots-clés : Architectures orientées service, composition de service, programmation chimique

1 Introduction

Service-oriented computing based on Web services represents currently one of the main drivers for the software industry [1]. A primary goal of Service Oriented Computing is to make a collection of software services accessible via standardized protocols, whose functionality can be automatically discovered and integrated into applications. Services can be composed, in order to build more complex services. Such compositions of individual Web services into an added-value, **composite Web service**, are usually represented as workflows, where vertices are individual web services, while edges model their interdependencies. Over the last years, despite the decentralized nature of the Internet, service infrastructures have been built upon highly centralized architectures. Data centers and Cloud computers act as servers centralizing the storage and processing required for the coordination of services and, more generally, of clients (users or businesses) of the Internet. Such architectures present several weaknesses. First, they generally suffer from poor scalability and low reliability, servers being potential processing and communication bottlenecks as well as single points of failure [2]. Also, they raise privacy issues, all data and control passing through central servers and repositories.

It becomes crucial to promote a decentralized vision of service infrastructures, as suggested in [3]. The benefits of a decentralized approach are manifold, with respect to drawbacks of centralized architectures. First, as the processing and data are distributed among a set of nodes, there is no single point of failure. No central server acts as a potential bottleneck, network traffic is reduced, and the approach is globally more scalable. Second, the direct and asynchronous fashion of communications (without the need for central coordination) brings better throughput and graceful degradation [4]. Finally, no server takes control over data and work, each node integrating a local workflow engine (referred to as *local-engine* in the following), and having only a partial view of the composition.

More specifically, the execution of a composite Web service relies on an engine responsible for coordinating data and control flows between involved services. Consider for instance a simple workflow W consisting of activity A performed at node a followed by activity B performed at node b. In a centralized vision, during the actual execution of W, the engine first invokes A by sending a message to node a, then waits for the result of A (sent by a), and finally invokes B. With a decentralized workflow engine, nodes a and b may communicate directly (rather than through a central coordinator node) to transfer data and control when necessary (*e.g.*, after A finishes).

Recently, nature-inspired metaphors have been shown to be of high interest for service coordination [5]. In this paper, we investigate a high-level model, based on a chemical metaphor, for a decentralized and dynamic coordination of composite Web services. More specifically, we rely on HOCL (*Higher-Order Chemical Language*) [6], formally representing the chemical programming paradigm. Within HOCL, a computation is seen as a set of reactions consuming some molecules while producing some new, inside a chemical solution. Reactions take place in a naturally decentralized and implicitly parallel way. HOCL provides the higher order: reaction rules can apply on other reaction rules. It has been recently shown that such a paradigm is well-suited to express service orchestration [7], and describe the enactment of workflows engine [8]. The proper investigation of this paper is to show that the chemi-

cal model is well-featured for underlying a decentralized execution of composite Web services. In the approach presented, both data and control flows of the workflow are distributed.

The paper is organized as follows. Section 2 presents the chemical programming paradigm in more details and the key points of its design making it suitable for our context. Section 3 shows the decentralized execution of a workflow through the chemical paradigm. Section 4 presents the related works. Finally, Section 5 draws some conclusions.

2 The Chemical Paradigm

The chemical paradigm is a programming style based on the chemical metaphor. Molecules (data) are floating in a chemical solution, and react according to reaction rules (program) to produce new molecules (resulting data). Reactions are conditional, and take place between some molecules satisfying a reaction condition. This process continues until no more reactions can be performed: the solution is said to be *inert*. Reactions take place in an implicitly parallel and autonomous way, and in a non-deterministic order.

Formally, the solution is represented by a multiset containing molecules, and rewriting/transformation rules specify the reactions between molecules. The Gamma model (General Abstract Model for Multiset Manipulation) [9] has been a pioneer work realizing the chemical paradigm. The multiset, which is the formal representation of the chemical solution, is the unique data structure in Gamma. The multiset works similarly to a shared address space on which multiple processors can operate independently.

In this paper, we use a chemical language enhanced with higher order, called HOCL (*Higher Order Chemical Language*) [6]. In HOCL, every entity is a molecule, including reaction rules. A program is a solution of molecules, that is to say, a multiset of atoms (A_1, ... ,A_n) which can be constants (integers, booleans, *etc.*), sub-solutions (denoted $\langle M_i \rangle$), or reaction rules.

Following the chemical paradigm, the execution of an HOCL program consists in applying reactions until the solution becomes inert. A reaction involves a reaction rule **one** P **by** M **if** V and a molecule N that satisfies the pattern P and the reaction condition V. The reaction consumes the rule and the molecule N, and produces M. The basic **one** P **by** M **C** reaction rule is one-shot: it disappears when it reacts. Its variant **replace** P **by** M **C** is n-shots: it is not consumed when it reacts. In the following, we use a more advanced syntax to declare and name molecules: **let** x = M_1 **in** M_2 is equivalent to M_2 where all occurrences of x are replaced by M_1. For instance, consider the following solution *MaxNumbers* which calculates the maximum value of a given set of numbers. The below example illustrates the expressiveness and higher order of HOCL, where reactions consume and/or produce other reaction rules.

$$\textbf{let } max = \textbf{ replace } x, y \textbf{ by } x \textbf{ if } x \geq y \textbf{ in } \langle 2, 3, 5, 8, 9, max \rangle$$

The rule max reacts with two integers x and y such that $x \geq y$ and replaces them by x (keep the integer with highest value). Initially, several reactions are possible: max can react with any couples of integers satisfying the condition: 2 and 3, 2 and 5, 8 and 9, *etc.* In order for the final solution to contain only the

result, we introduce a higher-order rule responsible to delete the max rule once the solution only contain the highest integer value. This introduces the need for sequentiality of events: we need to wait that all possible reactions between max and couples of integers took place before deleting the rule. Within the chemical model, the sequentiality is achieved through sub-solutions: to access a sub-solution, a rule has to wait for its inertia. In our example, this leads to the encapsulation of the solution:

$$\langle\langle 2, 3, 5, 8, 9, max\rangle,\ \textbf{one}\ \langle max = m, \omega\rangle\ \textbf{by}\ \omega\rangle$$

The m variable matches a rule named max, and ω matches all the remaining elements. One possible execution scenario within the sub-solution is the following (2 and 8, as well as 3 and 5, react first, producing the intermediate state):

$$\langle 2, 3, 5, 8, 9, max\rangle \rightarrow^* \langle 3, 5, 9, max\rangle \rightarrow^* \langle 9, max\rangle$$

Once the inertia is reached within the sub-solution, the one-shot rule can be triggered, extracting the result:

$$\langle\langle 9, max\rangle,\ \textbf{one}\ \langle max = m, \omega\rangle\ \textbf{by}\ \omega\rangle \rightarrow \langle 9\rangle$$

As we illustrated with a fine-grain example, HOCL provides the ability to express autonomic coordination of rules (without the need for any centralized control). The current state of a computation is represented by the solution, that constitutes an information system by itself. In other words, the multiset is a shared space providing the information required for dynamic coordination, such as a decentralized workflow execution.

3 Chemical Approach for Decentralized Workflow Execution

In this section, we describe our decentralized approach for workflow coordination based on a higher-order *chemical* framework, illustrating the adequacy of the chemical paradigm to execute composite Web services.

3.1 Architecture

As illustrated by Figure 1, the proposed architecture is composed by two core elements, namely the **Chemical Web Service (ChWS)** and a **multiset**. A ChWS is a chemical encapsulation of a Web service. It is co-responsible with other ChWSes of the coordination of the execution of workflows. Physically, ChWSes are hosted by some nodes, themselves interconnected by a network. A ChWS is basically composed by three elements, namely:

1. The encapsulation of a Web service invocation. The invocation to the effective possibly distant Web service, is encapsulated in a chemical expression readable by a chemical interpreter. The implementation of the Web service itself is not encapsulated, as shown in Figure 1.

2. A local storage space containing part of the multiset, *i.e.*, molecules and reaction rules constituting the information (data and control) related to the coordination of workflow execution.

3. An HOCL interpreter, working as the chemical local-engine executing the reactions according to molecules and reaction rules stored in the multiset, responsible for applying the defined workflow patterns and transferring data and control information to other ChWSes involved in a workflow.

The multiset is a space shared by all ChWSes, containing all information needed by ChWSes for a decentralized execution of a workflow, and in which each ChWS can operate independently. This information combines molecules representing data and ChWSes, rules representing control dependencies of the workflow, and rules for the coordination of its execution, as illustrated by Figure 2. Data and control dependencies of the workflow are defined beforehand using some workflow executable languages, like the well-known BPEL [10], an XML-based workflow language for Web services. The BPEL specification is translated into a chemical program, as detailed in Section 3.2. To coordinate the execution of the workflow, we also need some additional chemical rules, which are *generic*, *i.e.*, independent of a specific workflow. Section 3.3 focuses on these generic rules.

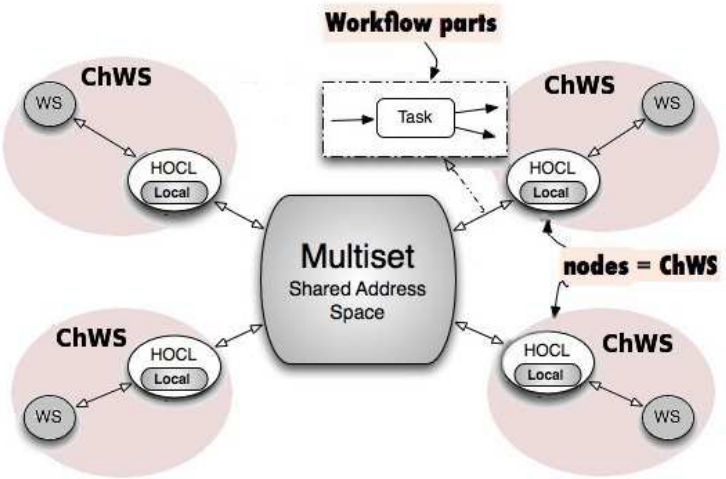

Figure 1: The proposed architecture.

The multiset shares some conceptual similarities with the Distributed Shared Memory (DSM) paradigm [11], developed in the area of distributed operating systems. DSM maps a globally unique logical memory address to a local physical memory slot, thus emulating a shared global space on top of a distributed memory platform. By analogy, multiset mirrors DSM's behavior by exposing molecules and reactions rules physically scattered across a set of ChWSes in a single shared space.

In other words, from a conceptual point of view (illustrated by Figure 1), ChWSes communicate through a unique global multiset containing all information needed by ChWSes to execute their part of a workflow. ChWSes exchange data and control dependencies through this multiset. In a classical centralized workflow architecture, the services themselves do not know these dependencies, as an engine manages all information and executes coordinates the whole execution.

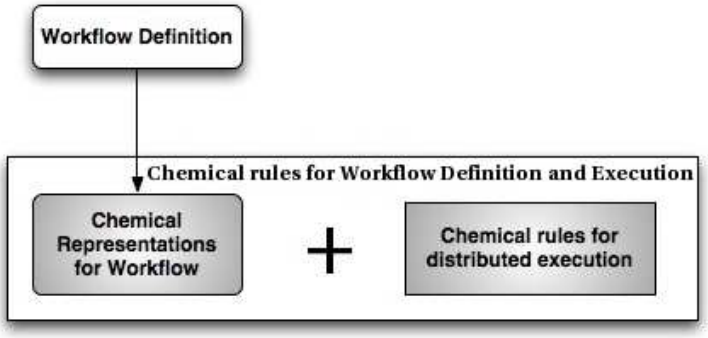

Figure 2: Chemical workflow.

From an implementation point of view, the multiset is physically distributed. While apparently, each ChWS only interacts with the multiset, physically, data and control information (molecules and reaction rules of the multiset) are effectively transferred between local storages of ChWSes. Put together, the molecules stored by ChWS form the multiset. Figure 3 summarizes these two points of view: the upper side shows the conceptual point of view where all ChWSes are *connected* through one multiset; the lower part shows the implementation point of view where all ChWSes are directly interconnected, the reactions and molecules being directly transferred from one ChWS to another one. Figure 3 provides a simple example where all ChWS are connected through a sequential workflow (modeled by arrows), but any workflow pattern could be applied, similarly.

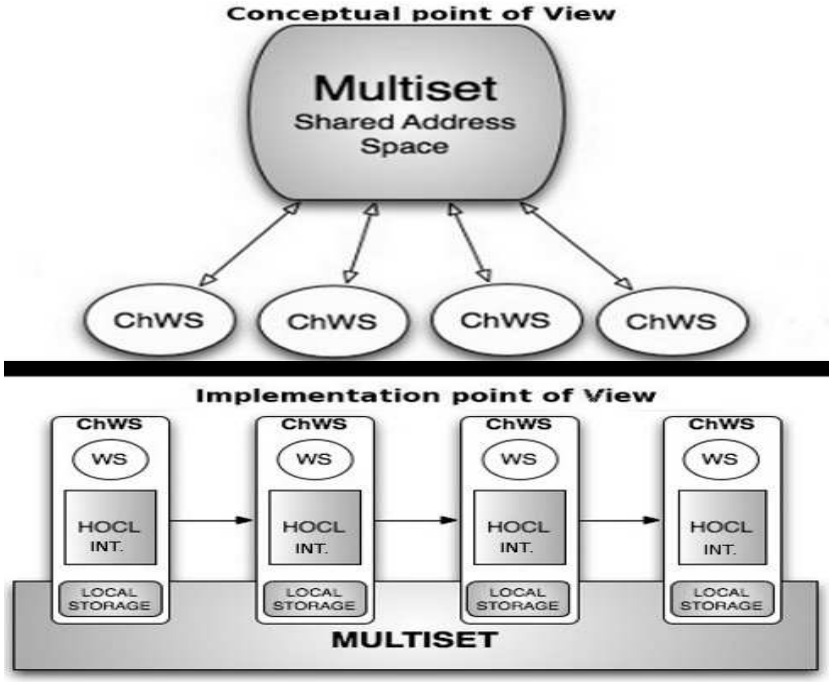

Figure 3: Different points of view.

3.2 Chemical Workflow Representation

In order to express all data and control dependencies of a workflow definition according to the chemical paradigm and to distribute the information among ChWSes, we use a series of chemical abstractions inspired by the work in [8]. These abstractions allow representing a workflow definition with the HOCL language. Such a representation is given in Algorithm 1.

As a chemical expression, the whole solution represents the multiset containing all information. The solution itself is composed of as many sub-solutions as ChWSes. Each sub-solution represents a ChWS with its data and control dependencies with other ChWSes within the workflow definition. More formally, a ChWS is one molecule of the form $ChWSi : \langle \ldots \rangle$. Algorithm 1 provides the general shape of such a solution. In the following, ChWS (in normal text) refers to the physical computational device realizing the Web service encapsulation, while $ChWS$ (in italics) refers to the molecule representing it inside the solution. Put differently, $ChWSi$ represents ChWSi in the solution.

Algorithm 1 General chemical workflow representation

```
1.01  ⟨ // Multiset (Solution)
1.02        ChWSi:⟨...⟩ // ChWS (Sub-solution)
1.03        ChWSi+1:⟨...⟩
1.04        ...
1.05        ChWSn:⟨...⟩
1.06  ⟩
```

We now illustrate such a representation with a simple workflow example composed of the four services shown on Figure 4. In this example, after S_1 completes, it invokes S_2 and S_3 in parallel. Once S_2 and S_3 have been completed, S_4 is invoked.

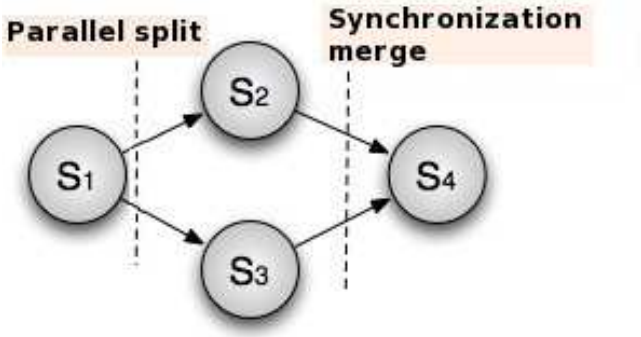

Figure 4: Simple workflow example.

The corresponding chemical representation for this workflow is presented in Figure 5. As we already mentioned, the solution contains as many sub-solutions as Web services. $ChWS1 : \langle \ldots \rangle$ to $ChWS4 : \langle \ldots \rangle$ represent ChWSes, and are sub-solutions within the global solution. The content of each $ChWSi$ expresses the data and control dependencies concerning it. To express the distribution

of information, we use a molecule of the form $Dest$: "$ChWSi$" with $ChWSi$ being the destination ChWS where some information needs to be transferred. For instance, we can see in the $ChWS1$ sub-solution that ChWS1 must transfer some information (the result of ChWS1) to ChWS2 and ChWS3 (refer to Line 2.01).

```
2.01  ChWS1:⟨Dest:"ChWS2",Dest:"ChWS3"⟩,
2.02  ChWS2:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S2:(R1):result⟩,
2.03  ChWS3:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S3:(R1):result⟩,
2.04  ChWS4:⟨replace ResultS2:R2, ResultS3:R3 by Call:S4:(R2):result⟩
```

Figure 5: Chemical workflow representation.

Let us focus on the details of the chemical representation of the workflow. As specified by the workflow shown in Figure 4, ChWS2 presents a data dependency in the sense that it requires the result of S_1 to start. In chemical words, the reaction rules triggering the call to S_2 needs a molecule $ResultS1 : R1$ that contains the result of S_1 to be performed (see the second part of line 2.02). The molecule produced by the reaction represents the call to S_2 and is expressed using a molecule of the general form *Call:Si:(Rj):result*, where R_j is the result of some service S_j and is used as an input parameter, S_i is the Web service invoked, and *result* represents the outcome of this service invocation. ChWS3 works similarly.

Occasionally, on a particular service, data and control dependencies may differ. Consider ChWS4. As specified by Figure 4, ChWS4 needs to wait until ChWS2 and ChWS3 have been completed. This constitutes a control dependency known as *synchronization merge*. However, as we can see in line 2.04, the service S_4 is invoked only on R_2 which is the result of S_2. This constitutes a data dependency. The $ChWS4$ sub-solution contains one reaction rule translating those dependencies in chemical language (see line 2.04): the presence of molecules $ResultS2 : R2$ and $ResultS3 : R3$ inside the $ChWS4$ sub-solution expresses the fulfillment of the control dependencies. The input $R2$ inside the $Call : S4 : R2 : result$ molecule expresses the data dependency. During the execution, as soon as $ResultS2 : R2$ and $ResultS3 : R3$ appear in the $ChWS4$ sub-solution, the local engine of ChWS4 will be able to perform the reaction that will produce a new molecule of the form $Call : S4 : R2 : result$ to call the effective service S_4 on the input $R2$.

To sum up, one reaction rule can express both control and data dependencies. In contrast with the previous synchronization merge pattern, the simple data dependencies are enough to express the simple parallel split pattern of S_1 with S_2 and S_3. Thanks to the implicit parallelism of the chemical execution model, the reaction rules inside $ChWS2$ and $ChWS3$ can be executed in parallel. Therefore, ChWS2 and ChWS3 will receive the result of S_1 from ChWS1 and the invocation of S_2 and S_3 will take place in parallel.

This fragment of HOCL code contains the chemical representation of a workflow. However, this representation is not enough to execute a workflow in a decentralized way. The next section presents additional rules to allow this decentralized execution.

3.3 Chemical Rules for Decentralized Execution

To ensure the execution of a chemical workflow, as described in the previous section, several additional chemical rules must be defined. These rules are called generic since they are independent from any chemical workflow representation. We identified the need for two sets of generic rules: those responsible for disseminating information among ChWSes and those in charge of executing the workflow step by step.

To transfer information among ChWSes, we defined a single rule shown in Algorithm 2. Rule **passInfo** transfers information such as molecules or reaction rules. This reaction rule takes place only with molecules of the form *ChWSj:⟨Pass:d:s⟩* and *ChWSi:⟨w⟩*. *ChWSj:⟨Pass:d:s⟩* is a molecule which represents the sub-solution corresponding to ChWSj, where a molecule *Pass:d:s* expresses what information stored at ChWSj (what molecules in $ChWSj$) has to be transferred to *d*. In this molecule, *d* represents the destination ChWS and *s* represents all molecules and reaction rules needed to continue the execution in *d* and subsequently. On the other hand, *ChWSi:⟨w⟩* is a molecule which represents the sub-solution corresponding to ChWSi (destination ChWS), *w* represents the current content of this sub-solution. Therefore, the molecule *s* will be transferred from sub-solution *ChWSj* to sub-solution *ChWSi* if the condition is satisfied. (See line 3.03.)

Algorithm 2 GR1 - Chemical rule transferring information

```
3.01  let passInfo = replace ChWSj:⟨Pass:idChWS:⟨s⟩ ⟩, ChWSi:⟨ w ⟩
3.02                       by ChWSi:⟨w, s⟩
3.03                       if (idChWS == ChWSi)
```

To execute the workflow, we defined two generic rules: **servInvocation** and **joinResult**. They are shown in Algorithm 3. The **servInvocation** reaction rule invokes a Web service S_i, where *Call:Si:params:result* represents the service invocation to S_i and *w* the current content of the *ChWSi* sub-solution. Its execution triggers the calling of service S_i (*i.e.*, the service associated with the ChWS) while incorporating the results of the service invocation within the solution. The **joinResult** reaction rule is used for preparing the tranfer of the results produced by the invocation encapsulated in ChWSi, to ChWSj. This will later trigger the execution of the **passInfo** rule as described previously. The **joinResult** rule captures one molecule of the form *ChWSi:⟨ResultSi:result, Dest:"ChWSj"⟩*. *ResultSi:result* is the result of S_i, while *Dest:"ChWSj"* comes from the chemical specification of the workflow such as the one presented in Figure 5.

Algorithm 3 GR2 - Chemical rules managing the execution

```
4.01  let servInvocation = replace ChWSi:⟨Call:Si:params:result, w⟩
4.02                       by ChWSi:⟨ResultSi:result, w⟩
4.03  let joinResult = replace ChWSi:⟨ResultSi:result, Dest:"ChWSj"⟩
4.04                   by ChWSi:⟨Pass:ChWSj:⟨ResultSi:result⟩ ⟩
```

Thanks to these reaction rules, the execution of a chemical workflow is decentralized since each ChWS is able to execute rules using its embedded HOCL interpreter, each ChWS achieving the coordination related to the service it encapsulates.

3.4 Execution

To better understand how our proposed approach works, we describe step by step the execution of the chemical workflow shown in Figure 5. These steps are listed in Figures 6 (steps 1-3), 7 (steps 4-7) and 8 (steps 8-10). The first step (in lines 5.01-5.05) corresponds to the initial state of the multiset: the generic rules[1] as described in Algorithms 2 and 3 and the chemical worfklow specification as shown in Figure 5. Recall that, thanks to the higher-order property, reaction rules react themselves with other molecules.

```
5.01   ⟨GR1, GR2,
5.02   ChWS1:⟨Dest:"ChWS2",Dest:"ChWS3", Call:S1:paramInput:result⟩,
5.03   ChWS2:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S2:(R1):result⟩,
5.04   ChWS3:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S3:(R1):result⟩,
5.05   ChWS4:⟨replace ResultS2:R2, ResultS3:R3 by Call:S4:(R2):result⟩ ⟩
                                  ↓
5.06   ⟨ GR1, GR2,
5.07   ChWS1:⟨Dest:"ChWS2",Dest:"ChWS3",ResutlS1:R1⟩,
5.08   ChWS2:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S2:(R1):result⟩,
5.09   ChWS3:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S3:(R1):result⟩,
5.10   ChWS4:⟨replace ResultS2:R2, ResultS3:R3 by Call:S4:(R2):result⟩ ⟩
                                  ↓
5.11   ⟨ GR1, GR2,
5.12   ChWS1:⟨Pass:ChWS2:⟨ResutlS1:R1⟩, ResultS1:R1,
5.13                  Pass:ChWS3:⟨ResutlS1:R1⟩ ⟩,
5.14   ChWS2:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S2:(R1):result⟩,
5.15   ChWS3:⟨Dest:"ChWS4", replace ResultS1:R1 by Call:S3:(R1):result⟩,
5.16   ChWS4:⟨replace ResultS2:R2, ResultS3:R3 by Call:S4:(R2):result⟩ ⟩
```

Figure 6: Workflow execution, steps 1-3.

Initially, the workflow is in the state described by step 1 of Figure 6. The only possible reaction is inside $ChWS1$. The **servInvocation** rule (part of GR2, and described in line 4.01 of Algorithm 3) is triggered by the HOCL interpreter of ChWS1, invoking S_1. The resulting molecule $ResultS1 : R1$, the result of S_1, is produced. Then, ChWS1's local engine makes this new molecule react with the **joinResult** rule, producing two new molecules for the distribution of this result to ChWS2 and ChWS3. (See lines 5.11 to 5.13). Finally, still through ChWS1's local engine, the **passInfo** reaction rule (GR1, described by Algorithm 2) can react with these molecules and transfer them to sub-solutions $ChWS2$ and $ChWS3$.

[1] For the sake of simplicity, the generic rules are represented by *GR1* and *GR2*.

6.01 ⟨**GR1**, **GR2**,
6.02 ChWS1:⟨ResultS1:R1⟩,
6.03 ChWS2:⟨Dest:"ChWS4", **ResultS1:R1**,
6.04 **replace** ResultS1:R1 **by** Call:S2:(R1):result⟩,
6.05 ChWS3:⟨Dest:"ChWS4", **ResultS1:R1**,
6.06 **replace** ResultS1:R1 **by** Call:S3:(R1):result⟩,
6.07 ChWS4:⟨**replace** ResultS2:R2, ResultS3:R3 **by** Call:S4:(R2):result⟩ ⟩

↓

6.08 ⟨**GR1**, **GR2**,
6.09 ChWS1:⟨ResultS1:R1⟩,
6.10 ChWS2:⟨Dest:"ChWS4", **Call:S2:(R1):result**⟩,
6.11 ChWS3:⟨Dest:"ChWS4", **Call:S3:(R1):result**⟩,
6.12 ChWS4:⟨**replace** ResultS2:R2, ResultS3:R3 **by** Call:S4:(R2):result⟩ ⟩

↓

6.13 ⟨**GR1**, **GR2**,
6.14 ChWS1:⟨ResultS1:R1 ⟩,
6.15 ChWS2:⟨Dest:"ChWS4", **ResultS2:R2**⟩,
6.16 ChWS3:⟨Dest:"ChWS4", **ResultS3:R3**⟩,
6.17 ChWS4:⟨**replace** ResultS2:R2, ResultS3:R3 **by** Call:S4:(R2):result⟩ ⟩

↓

6.18 ⟨**GR1**, **GR2**,
6.19 ChWS1:⟨ResultS1:R1⟩,
6.20 ChWS2:⟨**Pass:ChWS4:**⟨**ResultS2:R2**⟩, ResultS2:R2⟩,
6.21 ChWS3:⟨**Pass:ChWS4:**⟨**ResultS3:R3**⟩, ResultS3:R3⟩,
6.22 ChWS4:⟨**replace** ResultS2:R2, ResultS3:R3 **by** Call:S4:(R2):result⟩ ⟩

Figure 7: Workflow execution, steps 4-7.

The execution continues on Figure 7. ChWS2 and ChWS3 have received the result from ChWS1 (see lines 6.02 to 6.06). S_2 and S_3 can be invoked. ChWS2's and ChWS3's local engines, in parallel, trigger the reaction rules of the form (***replace*** *ResultS1:R1* ***by*** *Call:Si:Ri:result*). Two new molecules of the form *Call:Si:(R1):result* with $i = 2$ or 3, needed to invoke their respective Web services, are produced. (See lines 6.10 and 6.11.) Finally, the **servInvocation** and then the **joinResult** reaction rules are launched by local engines of ChWS2 and ChWS3, producing two new molecules in $ChWS2$ and $ChWS3$, respectively (see lines 6.13 to 6.22). These two molecules will be used for the transfer of the results of S_2 and S_3 to ChWS4.

The execution ends with steps in Figure 8[2], processed by ChWS4's local engine. Once the information from ChWS2 and ChWS3 is received by ChWS4, the reaction rule (***replace*** *ResultS2:R2, ResultS3:R3* ***by*** *Call:S4:(R2):result*) can react with results molecules to produce a new molecule for invoking service S_4 (see line 7.07). Finally, **servInvocation** reaction rule will take place producing the final result, *i.e.*, $ResultS4 : R4$.

With this example, we have shown that local engines within ChWSes are co-responsible for applying workflow patterns, invoking services, and propagating

[2]For space reasons, ChWSes have been put together on one line.

the results to other ChWSes. The coordination is achieved as reactions become possible, in an asynchronous and decentralized manner.

```
7.01    ⟨GR1, GR2,
7.02    ChWS1:⟨ResultS1:R1⟩, ChWS2:⟨ResultS2:R2⟩,
7.03    ChWS3:⟨ResultS3:R3⟩, ChWS4:⟨ResultS2:R2, ResultS3:R3,
7.04            replace ResultS2:R2, ResultS3:R3 by Call:S4:(R2):result⟩ ⟩
                                    ↓
7.05    ⟨GR1, GR2,
7.06    ChWS1:⟨ResultS1:R1⟩, ChWS2:⟨ResultS2:R2⟩,
7.07    ChWS3:⟨ResultS3:R3⟩, ChWS4:⟨Call:S4:(R2):result⟩ ⟩
                                    ↓
7.08    ⟨GR1, GR2,
7.09    ChWS1:⟨ResultS1:R1⟩, ChWS2:⟨ResultS2:R2⟩,
7.10    ChWS3:⟨ResultS3:R3⟩, ChWS4:⟨ResultS4:R4⟩ ⟩
```

Figure 8: Workflow execution, steps 8-10.

4 Related Works

There is a large number of works dealing with distributed execution of workflows. We can distinct two kinds of distributed coordination approach. In the first one, nodes interact directly. In the second one, they use a shared space for coordination.

Earlier works proposed decentralized architectures where nodes achieve the coordination of a workflow through the exchange of messages [12, 13]. Recent works [14, 15, 16] keep much interest in this type of coordination mechanism. In [14], the authors introduce service invocation triggers, a lightweight infrastructure that routes messages directly from a producing service to a consuming one, where each service invocation trigger corresponds to the invocation of a service. In [15], an engine is proposed based on a peer-to-peer application architecture wherein nodes (similar to local-engines) are distributed across multiple computer systems, but appear to the users as a single entity. These nodes collaborate, in order to execute a composite Web service with every node executing a part of it. Recently, a continuation-passing style, where information on the remainder of the execution is carried in messages, has been proposed [16]. Nodes interpret such messages and thus conduct the execution of services without consulting a centralized engine. However, this coordination mechanism implies a tight coupling of services in terms of spatial and temporal composition. Nodes need to know explicitly which other nodes they will potentially interact with, and when, to be active at the same time. In our approach, ChWSes exchange information by writing and reading the multiset. Then, the communication can be completely asynchronous since the multiset guarantees the persistentance of data and control dependencies. This makes our approach more relevant in a loosely-coupled services environment, and able to deal with dynamic changes in the workflow.

Another series of works rely on a shared space to exchange information between nodes of a decentralized architecture, more specifically called a *tu-*

plespace [17, 18]. Its origin can be found in the Linda coordination language [19] as a parallel programming extension for programming languages for the purpose of separating coordination logic from program logic. Linda builds upon the notion of a tuplespace, which is a piece of memory shared by all interacting parties. Using tuplespace for coordination, the execution of a part of a workflow within each node is triggered when tuples, matching the templates registered by the respective nodes, are present in the tuplespace. Thus, the templates a component uses to consume tuples, together with the tuples it produces, represent its coordination logic. In [17] and [18], approaches to replace a centralized BPEL engine by a set of distributed, loosely coupled, cooperating nodes, are presented. Both approaches present a coordination mechanism where the data is managed using a tuplespace and the control is driven by asynchronous messages exchanged between nodes. This message exchange pattern for the control is derived from a Petri net model of the workflow. In [18], the workflow definition is transformed into a set of activities, that are distributed by passing tokens in the Petri net. However, while in these works, the tuplespace is only used to store data information, our coordination mechanism stores both control and data information in the multiset, which is made possible by the use of the chemical execution model for the coordination of all data and control dependencies.

5 Conclusion

Most of today's approaches to the coordination of composite Web services are based on highly centralized architectures. Such systems present several drawbacks, mainly dealing with scalability, fault-tolerance, and privacy. In order to tackle these issues, it has become crucial to propose decentralized coordination mechanisms. However, current proposals for decentralized workflow coordination require tight coupling of services, and use workflow description languages that do not provide concepts for distributed workflow execution. In this paper, we have proposed a high-level coordination mechanism allowing a distributed execution of composite Web services, based on the chemical metaphor. Our chemical programming paradigm expresses parallelism and autonomic behaviors naturally using a higher-order chemical language. We have introduced the notion of the chemical Web service, which encapsulates a Web service. Through a shared multiset containing the information on both data and control dependencies needed for coordination, chemical Web services are co-responsible for carrying out the execution of a workflow in the composite services in which they appear. Spatial and temporal composition of services is achieved dynamically through this shared multiset. Their coordination is decentralized and distributed among individual Web service's chemical engine executing a part of the workflow. Finally, the approach shows the benefits of using the chemical paradigm for a decentralized coordination of composite Web service execution.

References

[1] Michael N. Huhns and Munindar P. Singh. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE*, 9(1):75–81, 2005.

[2] Gustavo Alonso, C. Mohan, Divyakant Agrawal, and Amr El Abbadi. Functionality and limitations of current workflow management systems. *IEEE Expert*, 12, 1997.

[3] Yun Yang. An architecture and the related mechanisms for web-based global cooperative teamwork support. *Int. Journal of Computing and Informatics*, 24, 2000.

[4] Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. *In Proceedings of the 13th International World Wide Web Conference, (WWW2004)*, pages 134–143, 2004.

[5] Mirko Viroli and Franco Zambonelli. A biochemical approach to adaptive service ecosystems. *Information Sciences*, pages 1–17, 2009.

[6] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Generalised multisets for chemical programming. *Mathematical Structures in Computer Science*, 16(4):557–580, 2006.

[7] Jean-Pierre Banâtre, Thierry Priol, and Yann Radenac. Chemical Programming of Future Service-oriented Architectures. *Journal of Software*, 4(7):738–746, 2009.

[8] Zsolt Németh, Christian Pérez, and Thierry Priol. Distributed workflow coordination: molecules and reactions. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006.

[9] Jean-Pierre Banâtre and Daniel Le Métayer. The gamma model and its discipline of programming. *Sci. Comput. Program.*, 15(1):55–77, 1990.

[10] OASIS Standard. Web services business process execution language, (WS-BPEL), Version 2.0. 2007.

[11] Jelica Protić, Milo Tomasević, and Veljko Milutinović. *Distributed shared memory.* John Wiley and Sons, 1998.

[12] Jun Yan, Yun Yang, and Gitesh Raikundalia. Enacting business processes in a decentralised environment with p2p-based workflow support. In *Advances in Web-Age Information Management*, pages 290–297. 2003.

[13] Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. In *Proceedings of the 19th conference on object-oriented programming, systems, languages, and applications*, pages 170–187. ACM, 2004.

[14] Walter Binder, Ion Constantinescu, and Boi Faltings. Decentralized orchestration of compositeweb services. In *Proceedings of the IEEE International Conference on Web Services*, pages 869–876. IEEE Computer Society, 2006.

[15] Rosa Anna Micillo, Salvatore Venticinque, Nicola Mazzocca, and Rocco Aversa. An agent-based approach for distributed execution of composite web services. In *IEEE International Workshops on Enabling Technologies*, pages 18–23, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[16] Weihai Yu. Consistent and decentralized orchestration of BPEL processes. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1583–1584, Honolulu, Hawaii, 2009. ACM.

[17] Paul A. Buhler and Jose M. Vidal. Enacting BPEL4WS specified workflows with multiagent systems. *In Proceedings of the Workshop on Web Services and Agent-Based Engineering*, 2004.

[18] Daniel Martin, Daniel Wutke, and Frank Leymann. A novel approach to decentralized workflow enactment. In *Enterprise Distributed Object Computing Conference, IEEE International*, pages 127–136, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[19] David Gelernter Yale University. Generative communication in linda. *ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1,*, pages 80–112, 1985.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399