Distributed Workflow Coordination: Molecules and Reactions

Zsolt Németh[1], Christian Pérez[2], Thierry Priol[2]

[1]MTA SZTAKI
Computer and Automation Research Institute
P.O. Box 63., H-1518 Hungary
zsnemeth@sztaki.hu

[2]IRISA
Campus Universitaire de Beaulieu
35042 Rennes, Cedex
Christian.Perez@irisa.fr,
Thierry.Priol@irisa.fr

Abstract

Workflow execution on large-scale heterogeneous distributed computing systems, such as Grids, requires a complex coordination. Activities of complex workflow patterns must be matched with entities of the computing system that possesses highly dynamic properties. We pinpoint the key concept of such workflow coordination as actions according to actual and local conditions – analogously to chemical reactions. Modeling workflow enactment as molecules and reactions, formalized in the nature inspired γ-calculus, yielded an autonomously evolving, distributed, decentralized coordination model that can adapt to a dynamically changing environment.

1 Introduction

According to the most general definition, a workflow is "the computerized facilitation or automation of a business process in whole or part" [12]. In the field of scientific computations it is understood in a more relaxed form as a collection of activities (computational tasks) that are processed in some order and where both data-flow and control-flow relationships are present. [9].

In compliance with the Workflow Reference Model [12], first, the application to be executed is decomposed into a number of independent activities that are related to each other by data and control dependency specified in the Process Definition. Subsequently, a Workflow Enactment Service instantiates and controls activities according to the Process Definition. Looking at more details ([8, 9]), Process Definition has two aspects. First the problem is transformed into the so called *Abstract Workflow (AW)* that expresses the logic of the problem to be solved yet, it does not contain any specific means how to be executed. This phase is related to the semantics and therefore, it is mostly up to the application writer's intention how to decompose a problem into separate activities and what relations are established. Next, *Abstract Workflow* is transformed into a *Concrete Workflow (CW)* where each logical entity in the AW is mapped onto physical entities (i.e. resources, services, processes etc.) that can enable the execution.

We take into consideration the execution of scientific workflows on large-scale heterogeneous distributed computing systems, like Grids. Such an environment is extent, highly dynamic with respect to performance and availability, and fault prone. We also consider the presence of advanced workflow patterns [1] that introduce some degree of uncertainty.

Generally, there is no consensus if the transformation of AW into CW belongs to the Process Design & Definition (build time) or the Workflow Enactment (run-time) phase – Process Definition in the reference model [12] may mean either of them on the boundary of build time and run-time activities. Some works tend to consider it as a build time activity by a static schedule or optimization of the mapping prior to run-time. In our view this transformation must be carried out at run-time and forms a central issue in workflow enactment that we define as a complex coordination between elements of the workflow the distributed system. Such a coordination comprises resource, service, data discovery and selection, handling control and data dependencies of activities, possibly fault detection and recovery.

We aimed at establishing a highly abstract declarative coordination model for executing workflows in large-scale heterogeneous distributed systems based on the following principles. (I) Resources can appear or disappear any time, abrupt changes may occur in the execution. Activities in the workflow should be able to react to the changes of the embedding environment. Therefore, enactment must be performed dynamically without any a priori decisions. (II) Workflow enactment should be able to make decisions on

1-4244-0054-6/06/$20.00

actual information. Information repositories contain data that are necessarily inconsistent with the actual state of resources. There cannot be a global and consistent view of all activities and resources assumed. (III) Workflow enactment should provide a higher level of autonomy and dynamic adaptation that at the same time also exhibits some degree of fault tolerance than most current approaches. (IV) Workflow coordination should be able to support advanced, complex control structures.

As in many cases, where the problem is complex and difficult to formalize, a nature metaphor may inspire a heuristic solution; in this case we have considered a chemical analogy. Activities of a workflow should be scheduled on resources autonomously, without any centralized control, yet observing certain rules just like particular molecules react according to laws of nature. We used γ-calculus to formalize the chemical metaphor and establish an inherently distributed declarative coordination framework. This paper presents the concept and some design alternatives – without actual technical details of implementation – by advanced examples where chemical coordination gives a sophisticated, yet elegantly simple solution to the requirements.

This paper is organized as follows. Section 2 gives the rationale behind our decision to use the chemical metaphor. Section 3 briefly introduces the γ-calculus. Section 4 describes the representation of resources as chemical solutions and then Section 5 shows how a workflow can be modeled as molecules and enacted as reactions. Further details and advanced scenarios of the proposed approach are illustrated by examples in Section 6. Section 7 presents related works close to our approach and finally, section 8 concludes the paper and outlines perspectives.

2 Why chemical reactions?

Chemical reactions take place without any control of each atom or molecule. Molecules react according to their chemical properties, actual local conditions and universal laws of nature. While reactions are unpredictable at the level of particular molecules, the overall properties of the matter are known and predictable based on chemical laws.

We envision workflow enactment in a similar, distributed and autonomous way using little or no a priori information but sensitively to changes in the infrastructure. There are resources and workflow activities that may be matched in numerous combinations. Instead of picking one possible enactment pattern in advance, as any a priori schedule would do, workflow enactment in our model is a process that evolves in time. Properties of resources and activities define the possible matchings just like chemical properties define the affinity of molecules to react. Actual conditions enable a certain enactment step just like they define the potential of a reaction. If properties and conditions are well defined, the outcome of the coordination is predictable and controllable.

In our coordination model resources and workflow activities form together a chemical solution. Within this solution, resources are passive molecules (precisely: sub-solutions), activities are active molecules that may capture others. The solution may also contain control molecules that influence reactions – sometimes their role is similar to catalysts. The entire coordination procedure is realized by the molecules and series of reactions, i.e. autonomously, independently from each other, concurrently, without any predefined pattern. The state of the computation is represented by the entire solution itself – it is a distributed information system by nature. The chemical model has two cornerstones: resource quantums, the way how resources are represented and γ-calculus, the modeling formalism.

The capacity of a resource is represented by quantums: a certain, guaranteed service on behalf of the resource. Activities may apply for such a quantum and if granted, they have the exclusive right to use the given quantum. They can be considered as a ticket for accessing a resource. Resource owners may offer all or parts of their capacities or services in quantums manifested as molecules. The strategy how quantums are established and how molecules are emitted is at the resource owners' discretion. For instance, a Linux cluster of 16 computing nodes may be transformed into quantums in any of the following forms according to a chosen strategy (details like network or storage are omitted here but can be added easily): 16 tickets as 1 node resource, 1 ticket as 16 node resource, 3 tickets as 4, 4, 8 node resources, respectively, and so on. Actual resource utilization strategies, validity and outreach of the quantums can be controlled by catalysts (see some examples in Section 6.) Resource quantums or resource molecules are the materialization of resource information and also serve as an access control mechanism; they ensure the validity of the information. Resources that are used exclusively are represented by a single molecule whereas resources that are shared by nature, e.g. networks, may emit multiple molecules into the coordination solution.

The vision of chemical enactment is specified as a concise coordination model using the γ-calculus. Both workflow structures (activities) and resources are formalized as γ-expressions that, at the same time, define their possible interactions, i.e. coordination semantics as well. Activities are represented as active molecules that can capture other, passive molecules like resources. The fact that all entities in the coordination framework are modeled using the same γ-formalism and the formalism defines the execution semantics as well is unique among all workflow coordination models.

3 The chemical paradigm and the γ-calculus

Gamma (General Abstract Model for Multiset Manipulation) [5] is a pioneer work among other languages realizing the so called chemical model - chemical in a sense that there is no concept of any centralized control, ordering, serialization rather, the computation is carried out in an indeterministic way according to local conditions.

The γ-calculus is a formal definition of the chemical paradigm that we applied in this work therefore, a very short introduction to γ-calculus is presented here based on [2] and [3]. The background and evolution of Gamma and the related models are introduced in [4, 5].

The fundamental data structure of the γ-calculus is the multiset M. γ-terms (molecules) are : variables x, γ-abstractions $\gamma\langle x\rangle.M$, multisets (M_1, M_2) and solutions $\langle M\rangle$ [2]. Note, that molecule is a synonym for all γ-terms. Since the word 'solution' has many meanings, if there is no ambiguity, solutions are be referred as molecules in this paper.

Juxtaposition of γ-terms is commutative $(M_1, M_2 \equiv M_2, M_1)$ and associative $(M_1, (M_2, M_3) \equiv (M_1, M_2), M_3)$. Commutativity and associativity are the properties that realize the 'Brownian-motion', i.e., the free distribution and unspecified reaction order among molecules that is a basic principle in the chemical paradigm [3].

γ-abstractions are the reactive molecules that can take other molecules or solutions and replace them by other ones by reduction. Due to the commutative and associative rules, the order of parameters is indifferent; molecules, solutions participating in the reaction are extracted by pattern matching – *any* of the matching ones may react. The semantics of a γ-reduction is

$$(\gamma\langle x\rangle.M), \langle N\rangle \rightarrow_\gamma M[x := N]$$

i.e., the two reacting terms on the left hand side are replaced by the body of the γ-abstraction where each free occurrence of variable x is replaced by parameter N if N is inert, or x is hidden in M, i.e., it only occurs as a solution $\langle x\rangle$ [3].

Besides the associativity and commutativity, reactions occur according to laws of (i) locality (also called as chemical) law [3]: if a reaction can occur, it will occur in the same way irrespectively to the environment; and (ii) solution (also called as membrane) law [3]: reactions can occur in nested solutions or in other words, solutions may contain sub-solutions.

A *conditional reaction* is a γ-abstraction of form $\gamma\langle x\rangle\lfloor C\rfloor.M$ that can be reduced only if C evaluates to true before the reaction [2].

Atomic capture adds the possibility of reacting with more than one molecule at a time. It is represented by an n-ary γ-abstraction term: $\gamma\langle x_1\rangle, \langle x_2\rangle, ...\langle x_n\rangle.M$ that can be reduced in a single atomic step if it can be matched by n appropriate terms [2].

The γ-calculus is a *higher order* model, where abstractions – just like any other molecules –, can be passed as parameters or yielded as a result of a reduction.

The γ-calculus shows some similarities with the λ-calculus. However, the λ-calculus is a sequential and deterministic model whereas, the γ-calculus is inherently parallel and nondeterministic.

4 Resources as chemical solutions

Resources (represented as quantums) are modeled as a chemical solution, i.e. sub-solutions within the global coordination solution. Attributes of the resource form molecules within the solution. The general form of a solution describing a resource quantum:

$\langle a_1 : v_1 a_2 : v_2 ... a_n : v_n\rangle$

where each $a_i : v_i$ forms an attribute-value pair. The exact number of attributes and their meaning is depending on the resource. Yet, there are some attributes that every resource should possess like identification, contact address, optionally further information about contact protocols, validity, security and so on. We do not specify any strict format at this level of abstraction, instead show many possibilities; nevertheless, these attributes must be used in a consistent way within an application. A certain implementation however, must have a well defined set of attributes.

The coordination solution may contain potentially hundreds or thousands of resource sub-solutions of various forms. A few examples:

- $r_1 = \langle id : R1,\ type : comp,\ proc : 16,\ OS : Linux,\ mem : 32,\ installed : equsolver, ...\rangle$ is a computational resource with 16 processors, Linux operating system, 32MB of memory and an equation solver package installed. Note, that a default unit is assumed for quantities: MB for memory, GB for disks, etc.
- $r_2 = \langle id : R2,\ type : comp,\ proc : 4,\ OS : SunOS,\ disk : 120,\ installed : statistics, network : N2...\rangle$ is another computational resource with 4 processor, SunOS, 120GB disk, a statistics package installed and accessing network N2.
- $r_3 = \langle id : R3,\ proc : 1,\ OS : Linux,\ available : 1200, ...\rangle$ – resource descriptions may contain attributes related to their use. In this case, for instance, $R3$ is available daily from 12.00.
- $r_4 = \langle id : R4,\ type : comp,\ OS : Linux,\ proc : 1,\ network : N2, owner : ownerid, ...\rangle$ – resource owner information may also be included.

- $r_5 = \langle id : N2,\ type : net,\ bandwidth : 10, ...\rangle$ – the network as a resource. Since it is not used exclusively, multiple occurrences of molecule r_5 is allowed. Note, that relevant resources must be represented only. For instance, if we assume that another network, N_1 provides a default connectivity, it is not necessary to include in the chemical solution or in the resource description.

5 Workflow as molecules and reactions

Workflow activities are modeled in the chemical solution as *active* molecules, i.e. a γ-abstraction in the form of $\gamma\, parameters.body$. A reaction means reducing the abstraction: capture matching molecules to the parameters, and substitute them in the body. The reaction replaces the captured molecules and the abstraction itself by the body with instantiated parameters.

The aim of coordination is to reduce dependent activities to single activities that are ready to be executed, i.e. all dependencies are satisfied. Each such single activity is treated as an atomic unit that is executed and some results are returned. The exact working behavior or the structure of the activity is irrelevant at the coordination level. An activity A ready to be executed is represented symbolically as:

$$\mathbf{execute}\ A\ \mathbf{on}\ resource(s)\ \mathbf{using}\ parameter(s)$$

where **execute** can be considered as a primitive procedure, i.e. it is not defined further in terms of γ-calculus. γ-reduction progresses until this normal form is reached. The entire chemical metaphor is aimed at *coordinating* the enactment of the workflow but not at *executing* its activities. Once a γ-expression has been reduced to this primitive, it is taken out of the chemical solution and passed to some external processes that realize the physical execution. The technical realization of the execution may vary since it depends on the physical environment (if it is a cluster, a Grid, a service oriented architecture, components, etc.) After the activity has been executed externally, results from the execution and the redeemed resources must be put back to the chemical solution, i.e. **execute** is reduced externally as:

$\mathbf{execute}\ A\ \mathbf{on}\ resource\ \mathbf{using}\ parameters$
$\rightarrow \langle A : result, ...\rangle, \langle id : resource, ...\rangle, ...$

Note that $result$ is a symbolic representation here. It may be anything: a number, a string, a complex structure, a pointer to a file, a pointer to any other data items or oppositely, holding no information at all just signaling the termination. Apart from the obligatory $activity : result$ pair they may contain other information as well related to the execution, e.g., error codes, number of iterations, resources that were used and so on.

Resource dependency. An activity requires resources to perform its task. We define the general form of resource dependency expressed in the γ-calculus as:

$$\gamma\langle a_1 : v_1, a_2 : v_2, ...a_n : v_n\rangle.\mathbf{execute}\ A\ \mathbf{on}\ v_i$$

where a_i is the mandatory identifier tag. The attributes specify a resource profile that will be found in the solution by pattern matching. This active molecule may capture *any* resource that has the specified profile, i.e., those solutions where $attribute : values$ pairs can be matched. This general form may involve variables and the universal matching symbol, ω, as it is introduced in the following simple examples.

For instance, activity A_1 requiring a computing node with 1 processor and Linux OS may be specified as:

$$\gamma\langle id : r, proc : 1, OS : Linux, \omega\rangle\ .\ \mathbf{execute}\ A_1\ \mathbf{on}\ r$$

where ω matches anything, i.e. rest of the resource profile is irrelevant. Without specifying ω, it would match only a resource that has no other tags but id and $proc$.

Taking resources from Section 4 as an example, pattern matching may be successful in two cases thus, r may be bound to either $R3$ or $R4$ and the expression reduced to
$\rightarrow \mathbf{execute}\ A_1\ \mathbf{on}\ R3$ or
$\rightarrow \mathbf{execute}\ A_1\ \mathbf{on}\ R4$
Which one is actually chosen – it is up to chance in the chemical model where particular reactions are unpredictable. Nevertheless, the basic chemical paradigm where Brownian motion assures random choice can be augmented with further nature laws. For instance, adding *gravity* to the model could be represented by ordering the molecules and choosing the first or last in these cases.

Data/control dependency. An activity may depend on the result of other activities (data dependency) or simply triggered by the termination of another activity (control dependency).

If activity A_i depends on another activity A_j, i.e., they are executed in *sequence* : $A_j \rightarrow A_i$, the corresponding γ-expression is:

$$\gamma\langle A_j : x, \omega\rangle.\mathbf{execute}\ A_i\mathbf{using}\ x$$

i.e. the execution of A_i requires a result molecule (precisely: sub-solution) that is emitted by A_j.

If activity A_l depends on n other activities A_{s_i}, i.e., realizes a *synchronization*, the active molecule captures n result molecules in an atomic step:

$\gamma\ \langle A_{s_1} : x_1, \omega_1\rangle, \langle A_{s_2} : x_2, \omega_2\rangle, ...\langle A_{s_n} : x_n, \omega_n\rangle.$
$\mathbf{execute}\ A_i\mathbf{using}\ x_1, x_2, ...x_n$

In case of a *parallel-split*, i.e. there are m activities depending on A_l, the result molecule must by multiplied m times by a γ-abstraction added to the body of A_l:

$(\gamma\ \langle\langle A_l : x, \omega\rangle\rangle.\langle A_l : x, \omega\rangle, \langle A_l : x, \omega\rangle, ...\langle A_l : x, \omega\rangle),$
$\langle \mathbf{execute}\, A_l\rangle$

Note, that the execution of A_l takes place in a solution ($\langle \mathbf{execute}\, A_l\rangle$) therefore, its result will be a sub-solution within the solution ($\langle\langle A_l : r_{A_l}\rangle\rangle$) and reducing further:

$\rightarrow (\gamma\ \langle\langle A_l : x, \omega\rangle\rangle.\langle A_l : x, \omega\rangle, \langle A_l : x, \omega\rangle, ...\langle A_l : x, \omega\rangle),$
$\langle\langle A_l : r_{A_l}\rangle\rangle$
$\rightarrow \langle A_l : r_{A_l}, \omega\rangle, \langle A_l : r_{A_l}, \omega\rangle, ...\langle A_l : r_{A_l}, \omega\rangle$

the result molecules are multiplied. This construct "hides" the result molecule ($\langle A_l : r_{A_l}\rangle$) otherwise, it could react with any of the depending activities *before* the multiplication takes place.

These three γ-expressions realize *sequence*, *synchronized merge* and *parallel split* constructs – many workflow management systems do not go beyond these ones.

Informally we introduce further, advanced constructs that are realized as a chemical reaction. For exact details of their formal γ-definition see [13]. In case of **split** there is number of potential follow-up activities but from them *exactly p* are activated. **Exclusive choice**, as a special case for split, where from all potential activities *one and only one* may be activated after termination of A_l. By combining it with conditional, **explicit choice** can be realized. In case of **p out of n merge** there are n antecedent activities that may produce input for A_l but the termination of *exactly* p, may activate A_l. $p = 1$ means that the termination of *any* result may activate A_l. A **structured loop** can be constructed from a merge and a conditional structure furthermore, the core of the loop is added to the solution dynamically in each iteration. Some of these constructs are illustrated in the next section.

Resource and data/control dependencies do not differ in the sense that they represent conditions that all must be met before the activity can proceed. Thus, if data/control and resource dependencies must be satisfied simultaneously, the active molecule performs an atomic capture:

$\gamma\ (\langle A_j : x, \omega_a\rangle, \langle id : r, R, \omega_r\rangle).\mathbf{execute}\ A_i\ \mathbf{on}\ r\ \mathbf{using}\ x$

For instance, if activity A_2 is dependent on the result of A_1 requiring a computing node with 1 processors may be specified as:

$\gamma\ \langle id : r, proc : 1, \omega_r\rangle\langle A_1 : x, \omega_a\rangle\ .$
$\mathbf{execute}\ A_2\ \mathbf{on}\ r\ \mathbf{using}\ x$

After A_1 has finished and returned, e.g. $\langle A_1 : 42, ...\rangle$, where 42 is a fictitious result of some calculations, reduction may go on in one of two different directions depending on the result of the pattern matching as (resources taken from Section 4):

$\rightarrow \mathbf{execute}\ A_2\ \mathbf{on}\ R3\ \mathbf{using}\ 42$ or
$\rightarrow \mathbf{execute}\ A_2\ \mathbf{on}\ R4\ \mathbf{using}\ 42$
As it can be seen, in this case A_2 can be executed if the specified resources have been found *and* A_1 has finished.

6 Advanced examples of the chemical model

Resource and data dependency. Obviously, both data/control and resource dependencies must be satisfied in order to execute the activity. Yet, the reaction can be enabled in different ways by structuring the capture of molecules.

The capture of the resource molecule can be postponed until activity A_i is enabled according to data/control flow rules (R is a symbolic notation for a resource profile):

$\gamma\ \langle A_j : x, \omega_a\rangle.$
$(\gamma\langle id : r, R, \omega_r\rangle.\mathbf{execute}\ A_i\ \mathbf{on}\ r\ \mathbf{using}\ x)$

This construct is effective when it is not known if a particular activity will be ever enabled (e.g. it is part of a conditional branch). First it captures the result, e.g. $\langle A_j : 42, ...\rangle$ and reduces to

$\rightarrow \gamma\langle id : r, R, \omega_r\rangle.\mathbf{execute}\ A_i\ \mathbf{on}\ r\ \mathbf{using}\ 42$

enabling the reaction with a resource molecule of profile R.

Conversely, the capture order can be changed expressing that an appropriate resource must be found before data/control flow may enable the activity. It may gain efficiency since the search for a resource may overlap with computations that will yield enabling A_i.

$\gamma\ \langle id : r, R, \omega_r\rangle.$
$(\gamma\langle A_j : x, \omega_a\rangle.\mathbf{execute}\ A_i\ \mathbf{on}\ r\ \mathbf{using}\ x)$

Constrained resource matching. Everyday examples require and the chemical paradigm makes it possible to easily express dependencies or constraints among resources as well. For instance, activity A requires simultaneously a computing node with disk space of at least 30G and a computing node with at least 128M memory so that the OS of the two nodes are the same:

$\gamma\ \langle id : r_1, OS : x, disk : y, \omega_1\rangle,$
$\langle id : r_2, OS : x, memory : z, \omega_2\rangle \lfloor y > 30, z > 128\rfloor.$
$\mathbf{execute}\ A\ \mathbf{on}\ r_1, r_2$

Constraints may exist between different execution stages as well. Let us assume the following requirements: A_1 needs a node with 1 processor, A_2 needs a node with 4 processors so that there must be a network connection of given bandwidth between them to transfer resulted files. The active molecule captures three resource molecules in a single reaction. The molecule corresponding to the network is a catalyst in this case: it must be present with given characteristics but it is not reserved by the activities:

$\gamma\,(\langle id : r1, proc : 1, network : n, \omega_{r_1}\rangle,$
$\quad\langle id : r2, proc : 4, netowrk : n, \omega_{r_2}\rangle,$
$\quad\langle id : n, bandwidth : 10, \omega_n\rangle)\,.$
$(\mathbf{execute}\, A_1\, \mathbf{on}\, r1,\ (\gamma\langle A_1 : x, \omega\rangle\,.\,\mathbf{execute}\, A_2\, \mathbf{on}\, r2\, \mathbf{using}\, x))$

Let us assume, pattern matching yielded $r1 = R4$ and $r2 = R2$ and $n = N2$ (see Section 4) and hence, the expression is reduced in the following order:

$\mathbf{execute}\ A_1\, \mathbf{on}\, R4, \gamma\langle A_1 : x, \omega\rangle\,.\,\mathbf{execute}\, A_2\, \mathbf{on}\, R2\, \mathbf{using}\, x$
$\rightarrow \langle A_1 : 23, ...\rangle,\ \gamma\langle A_1 : x, \omega\rangle\,.\,\mathbf{execute}\, A_2\, \mathbf{on}\, R2\, \mathbf{using}\, x$
$\rightarrow \mathbf{execute}\, A_2\, \mathbf{on}\, R2\, \mathbf{using}\, 23$
$\rightarrow \langle A_2 : 42, ...\rangle$

Another approach may take the advantage that a result molecule can contain any attributes as long as it is valid (not conflicting) and used in a consistent way in a single application. Thus, the utilized resource profile(s) can be embedded as a named sub-solution in the result molecule. In this example A_1 is executed on a resource with at most 4 processors and subsequent activities must have the same number of processors:

$\gamma\,\langle id : r1, type : comp, proc : p, \omega\rangle\ \lfloor p \leq 4\rfloor.$
$\mathbf{execute}\, A_1\, \mathbf{on}\, r1$

and A_2 can extract the constraint from the result of A_1:

$\gamma\,(\langle id : r2, proc : p, \omega_r\rangle,$
$\quad\langle A_1 : x, used\_resource = \langle proc : p, \omega_{ur}\rangle, \omega_a\rangle).$
$\mathbf{execute}\, A_2\, \mathbf{on}\, r2\, \mathbf{using}\, x$

that is, the resource to be reserved must have p processors. Note, that p is unknown prior to run-time.

Resource control. Resource tickets should be canceled whenever the resource is not available any longer either according to the owner's decision, to an error or to changes in the management policy. In terms of the chemical metaphor: a neutralizing agent must be added to the global solution:

$\gamma\langle a_1 : v_1, a_2 : v_2, ...a_n : v_n\rangle.$

that matches the specified resource molecule and removes it, i.e. replaces it to nil.

Further examples: a resource owner may want to change some attributes (in this case, for instance, $a_2 : v_2$ to $a_2 : v_2'$) of a resource by a molecule: $\gamma\langle a_1 : v_1, a_2 : v_2, ...a_n : v_n\rangle.\langle a_1 : v_1, a_2 : v_2', ...a_n : v_n\rangle$ Even further, a complete resource sharing policy may be changed by capturing and re-issuing the resource molecules. For instance, a resource owner may decide to withdraw two 4-node resources and offer one 8-node instead. In this case the following control molecule is added:

$\gamma\,\langle id : R_1,\ type : comp,\ proc : 4,\ owner : self, ...\omega_1\rangle,$
$\quad\langle id : R_2,\ type : comp,\ proc : 4,\ owner : self, ...\omega_2\rangle.$
$\langle id : R,\ type : comp,\ proc : 8, ...\rangle$

These molecules provide the logical control of changing the molecules in an abstract way. The realization of removing or changing a resource molecule obviously requires authorization procedures at the level of technical implementation.

Fault tolerance. The topic of fault tolerance is far beyond the scope of this paper. The goal of this section is to show by a few examples that the chemical framework is able to provide constructs for such purposes.

A failed activity A (or an agent detecting the failure, on behalf of A) may return a result molecule $\langle A : result, ...\rangle$ where $result$ contains some sort of an **error code**. Subsequently an activity B expecting the termination of A may include routines to handle the situation. If necessary, **redundant execution** is also possible: by using split and merge structures some activities may be initiated multiple times. If any of them succeeds, the rest of them are omitted. Using the loop structure it is possible to **retry** some critical activities that might fail.

A more sophisticated **rollback structure** can also be easily defined using γ-calculus. An activity puts a specific molecule into the solution upon its reaction. The molecule can reinstall the activity if necessary. For instance, activity A requires resource profile R, expects results from A_i and A_j and in case of a certain failure it should be re-executed on another resource. In this case the reaction reduces to the single activity and also puts an active molecule that in case of an error re-schedules the resource and executes A again:

$\gamma\,\langle A_i : x, \omega_{a1}\rangle,$
$\quad\langle A_j : y, \omega_{a2}\rangle,$
$\quad\langle id : r, R, \omega_r\rangle.$
$(\mathbf{execute}\, A\, \mathbf{on}\, r\, \mathbf{using}\, x, y,$
$(\gamma\langle A : error, \omega_{err}\rangle, \langle id : r', R, \omega_{r'}, \rangle.$
$\quad\mathbf{execute}\, A\, \mathbf{on}\, r'\, \mathbf{using}\, x, y))$

Obviously, instead of re-executing A, some other activities may be instantiated as well, like compensation; the resource scheduling policy may be changed and so on.

Complex workflow structures. Finally, a workflow structure is shown to demonstrate that advanced patterns can be realized easily. The workflow in Figure 1 is to be executed with the following assumptions: (i) Activities do not have any specific resource needs (therefore resource specifications are oversimplified, symbolic R), (ii) there must be a specific link between A_1, A_2 and A_3 (iii) activities between A_1 and A_4 are executed in a loop, A_5 is executed after the loop. Loop condition is tested on the result of A_1 and (iv) activity A_4 synchronizes.

Activities are expressed by the following molecules. The first activity can be triggered in two cases: initially (no

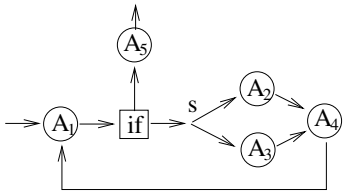

Figure 1. An example workflow

dependence) and re-entering the loop (dependent on A_4. Therefore, A_1 has two active molecules:

$a_{1_a} = \gamma\langle id : r, R, \omega\rangle \,.\, \mathbf{execute}\, A_1\, \mathbf{on}\, r$
$a_{1_b} = \gamma\langle id : r, R, \omega\rangle, \langle A_4 : x\rangle \,.\, \mathbf{execute}\, A_1\, \mathbf{on}\, r\, \mathbf{using}\, x$

S is a virtual activity introduced for a simpler representation of the conditional. If loop condition is true, virtual activity S is enabled, copies results of A_1 and puts into the solution the loop core: a_4, a_{1_b} and itself.

$s = \gamma\langle A_1 : x, \omega\rangle \lfloor C(x)\rfloor \,.\, \langle S : x, \omega\rangle,\ \langle S : x, \omega\rangle,\ a_4,\ a_{1b},\ s$

The copied results trigger A_2 and A_3

$a_2 = \gamma\langle id : r, R, \omega\rangle, \langle S : x, \omega\rangle \,.\, \mathbf{execute}\, A_2\, \mathbf{on}\, r\, \mathbf{using}\, x$
$a_3 = \gamma\langle id : r, R, \omega\rangle, \langle S : x, \omega\rangle \,.\, \mathbf{execute}\, A_3\, \mathbf{on}\, r\, \mathbf{using}\, x$

A_4 is enabled if both results are present in the solution:

$a_4 = \gamma\langle id : r, R, \omega\rangle, \langle A_2 : x, \omega\rangle, \langle A_3 : y, \omega\rangle \,.$
$\quad \mathbf{execute}\, A_4\, \mathbf{on}\, r\, \mathbf{using}\, x, y$

Finally, A_5 is triggered by the result of A_1 if the repeat condition does not hold:

$a_5 = \gamma\langle id : r, R, \omega\rangle, \langle A_1 : x, \omega\rangle \lfloor \neg C(x)\rfloor \,.$
$\quad \mathbf{execute}\, A_5\, \mathbf{on}\, r\, \mathbf{using}\, x$

To ensure resource constraints as well, a_1, a_2 and a_3 are merged together:

$$\begin{aligned} a_{1_a} = \gamma\, &\langle id : r1, network : n, R, \omega_1\rangle, \\ &\langle id : r2, network : n, R, \omega_2\rangle, \\ &\langle id : r3, network : n, R\omega_3\rangle, \\ &\langle id : n, type : network, \omega_n\rangle \,. \\ &(\mathbf{execute}\, A_1\, \mathbf{on}\, r1, \\ &\gamma\,\langle S : x, \omega_{s_1}\rangle \,.\, \mathbf{execute}\, A_2\, \mathbf{on}\, r2\, \mathbf{using}\, x, \\ &\gamma\,\langle S : y, \omega_{s_2}\rangle \,.\, \mathbf{execute}\, A_3\, \mathbf{on}\, r3\, \mathbf{using}\, y) \end{aligned}$$

$$\begin{aligned} a_{1_b} = \gamma\, &\langle id : r1, network : n, R, \omega_1\rangle,\ \langle A_4 : x, \omega_a\rangle, \\ &\langle id : r2, network : n, R, \omega_2\rangle, \\ &\langle id : r3, network : n, R, \omega_3\rangle, \\ &\langle id : n, type : network, \omega_n\rangle. \\ &(\mathbf{execute}\, A_1\, \mathbf{on}\, r1\, \mathbf{using}\, x, \\ &\gamma\,\langle S : y, \omega_{s_1}\rangle \,.\, \mathbf{execute}\, A_2\, \mathbf{on}\, r2\, \mathbf{using}\, y, \\ &\gamma\,\langle S : z, \omega_{s_2}\rangle \,.\, \mathbf{execute}\, A_3\, \mathbf{on}\, r3\, \mathbf{using}\, z) \end{aligned}$$

At the beginning, the coordination solution contains a_{1_a}, s and a_5. Other activities are put into the solution dynamically by S each time the loop is re-entered. A notable example of the dynamic behaviour of the chemical model.

7 Related works

There are numerous papers on workflow management yet, the enactment itself is rarely targeted directly. In this section some of those enactment models are listed that are (partly) dynamic, address a higher level coordination model and are closer to our approach in some sense.

Deelman et al. [8] propose applying AI techniques for planning an abstract workflow for applications where many large files are transferred between components. The mapping of abstract workflow onto concrete workflow is prior to execution and is based on static information. In a later paper [7] this policy is relaxed and a semi-dynamic mapping is introduced where the workflow is divided into smaller workflows and such partial workflows are mapped at one time. The work presented in [11] introduces a Petri-net based model. It has dynamic features: Petri-nets can be refined and on-the-fly resource mapping is also present. It supports more workflow constructs than usual and provides some degree of fault tolerance. Petri-nets also make a step towards a formal enactment model. Condor DAGMan [6] provides an on-the-fly dynamic mapping with fault tolerant features. Its functionality is similar to ours, but the approach is however, different. Condor is aimed at providing a high throughput computing environment by maximizing processor efficiency. Its structure and the scheduling policy is fixed, the supported workflow constructs are limited. The Workflow Enactment Engine (WFEE) [14] is probably the closest to our work. It is a distributed event-driven workflow enactment engine that realizes fully dynamic, just-in-time resource mapping. It utilizes a tuple space to maintain the state of the computation where tuples are related to events. WFEE makes its scheduling decisions based on stored information.

8 Future work & Conclusion

The chemical coordination model is highly abstract yet, it is not very far from a real-life implementation. An active molecule $\gamma P\lfloor C\rfloor.M$ may be interpreted Gamma-style [2, 3, 5] as

$$\mathbf{replace}\, P\, \mathbf{by}\, M\, \mathbf{if}\, C$$

that brings it close to $if\text{-}then$ rules, for which advanced pattern matching algorithms of production systems, like RETE [10], an "efficient method for comparing a large collection of patterns to a large collection of objects" can serve as a foundation for implementation.

We project the coordination system as a network of distributed workflow engines above the resources. Engines keep reacting the molecules, i.e., interpreting the γ-terms until some of them is reduced to an **execute** primitive that

is an exit point from the coordination level. In this case the **execute** primitive is removed from the solution and passed to a process that realizes all the procedures related to physical enactment corresponding to the activity and the parameters. The same process receives results from physical entities, turns them into molecules and puts them back to the coordination level. Similarly, resource quantums or error reports can be submitted via appropriate interfaces. In such a way the coordination is independent from any actual technical realization.

In this paper we introduced a workflow coordination model based on a chemical metaphor. Most of current approaches, although technically sophisticated, pose some restrictions on the workflow structure or dynamic behaviour, are aimed at some particular problems and quite often, lack a high-level model. We aimed at establishing a general abstract model and envisage workflow coordination as an autonomous process evolving in time according to dependencies and requirements but not bound to any a priori decision or static pattern. In this sense it shows considerable similarities with chemical reactions and thus, the work is based on a chemical metaphor. In our model resources, workflow activities and control information form a chemical matter and enactment progresses by reactions according to actual and local conditions. The informal concept of chemical workflow enactment is turned into a coordination model by applying the γ-calculus. The γ-calculus provides a mathematically well founded declarative formalism for describing arbitrarily complex workflows and advanced coordination strategies; execution semantics is given without further definitions.

The work presented in the paper is not a solution for workflow enactment; many issues of workflow scheduling are not addressed directly. Rather, the model is a highly abstract framework where various advanced enactment strategies can be specified. Since it is a high-level model and thus, independent from technical details, the principles expressed in the γ-calculus can be turned into implementation in several ways.

9 Acknowledgments

The research presented in this paper was carried out while Zs. Németh was hosted by IRISA-INRIA, Rennes, supported by the ERCIM Fellowship programme and partially supported by CoreGrid IST-2002-004265. Authors express their gratitude to Yann Radenac and Jean-Pierre Bânatre for their help with the γ-calculus.

References

[1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros: Workflow Patterns. BETA Working Paper Series, WP 47, Eindhoven University of Technology, Eindhoven, 2000.

[2] J.-P. Banâtre, P. Fradet, Y. Radenac: Principles of Chemical Programming. Fifth International Workshop on Rule-Based Programming (RULE'04), Electronic Notes in Theoretical Computer Science.

[3] J.-P. Banâtre, Y. Radenac, P. Fradet: Chemical Specification of Autonomic Systems. Proc. of the 13th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering (IASSE'04)

[4] J.-P. Banâtre, P. Fradet, D. Le Métayer: Gamma and the Chemical Reaction Model: Fifteen Years After. Multiset Processing, LNCS 2235, Springer 2001, pp. 17-44.

[5] J.-P. Banâtre, D. Le Métayer: Programming by Multiset Transformation. Communications of the ACM, Vol. 36, No. 1, January 1993, pp. 98-111.

[6] Condor DAGman http://www.cs.wisc.edu/condor/dagman/

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, M. Livny: Pegasus: Mapping Scientific Workflows onto the Grid. Across Grids Conference 2004, Nicosia, Cyprus, 2004

[8] E. Deelman, J.Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda: Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing, Vol. 1, No. 1, pp. 25-39, 2003

[9] T. Fahringer, J. Qin, S. Hainzer: Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, May 9-12, 2005.

[10] C. L. Forgy:Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence Vol. 19, No. 1, Sept. 1982, pp. 17-37.

[11] A. Hoheisel, U. Der: Dynamic Workflows for Grid Applications. Proceedings of the 3rd Cracow Grid Workshop, 2003.

[12] D. Hollingsworth: The Workflow Reference Model 19.Jan.95. Workflow Management Coalition, Doc. No. TC00-1003, 1995.

[13] Zs. Németh, C. Pérez, T. Priol: Workflow Enactment Based on a Chemical Metaphor. 3rd IEEE International Conference on Software Engineering and Formal Methods, SEFM 2005, Koblenz, Germany, IEEE Computer Society Press.

[14] J. Yu and R. Buyya: A Novel Architecture for Realizing Grid Workflow using Tuple Spaces, Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004, Nov. 8, 2004, Pittsburgh, USA), IEEE Computer Society Press, Los Alamitos, CA, USA.