FORMALIZATION OF AN RNA-INSPIRED MIDDLEWARE FOR COMPLEX SMART OBJECT FEDERATION SCENARIOS

Jérémie Julia[1], Yuzuru Tanaka[1] and Nicolas Spyratos[2]
[1]*Meme Media Laboratory, Hokkaido University, Sapporo, Japan*
[2]*Laboratoire de Recherche en Informatique, Université Paris-Sud 11, Paris, France*

Keywords: Ubiquitous Computing, Pervasive Computing, Service Federation, Smart Object.

Abstract: This paper proposes a new approach to deal with the smart objects or smart mobile devices by proposing a middleware framework inspired by RNA mechanisms in molecular biology. Our framework represents complex application scenarios of autonomic federation of smart objects as catalytic reaction networks. Each catalytic reaction is modeled as an RNA expressions from a DNA. Our framework is capable of dealing not only with the two stereotyped scenarios of ubiquitous computing, i.e. location-transparent service continuation and location-and/or situation-aware service provision, but also with much more complex federations scenarios of smart objects.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Nowadays, our environment is filled with computing devices such as RFID tags, chips with sensors and/or actuators, smart phones, PDAs, intelligent electronic appliances, embedded computers, and access points to network servers: computing devices called smart objects (Tanaka, 2008)(Tanaka, 2010). However, we are using only a tiny portion of their potentialities since we generally do not dynamically and/or flexibly connect them to create a federated complex ubiquitous environment. Ubiquitous computing has been addressing just two stereotyped kinds of scenarios: location-transparent service continuation, and location- and/or situation-aware service provision. It was pointed out that the absence of formal model is an obstacle preventing us from finding new kinds of scenarios (Milner, 2004)(Henricksen et al., 2002). Some researchers are trying to extend the application target of ubiquitous computing by using formal computation models of process calculi, e.g., Bigraphical Reactive System (Milner, 2001). These trials mainly focus on mathematical description and inference of the behavior of a set of mobile objects, but not those of the dynamically changing interconnection structures among mobile physical objects based on abstract description of their interfaces. That is why they are not sufficient to describe new kinds of scenarios. A lot of other studies have been conducted on the mathematical description of network topology for network reconfiguration and rerouting. These studies focused on physical connectivity among nodes, but not on their logical or functional connectivity. Thus they cannot describe application frameworks. These models are not sufficient to provide a basis for application frameworks that go beyond the existing stereotyped scenarios. We believe that any essential extension beyond these stereotypes will require formal modeling that can describe more complex federation scenarios among smart objects.

In conventional research studies, federation mechanisms were based on the matching of a service requesting message with a service providing message. This matching was made by a centralized-and-repository lookup service (as in the case of Linda(Gelernter, 1985)), or by a distributed repository-and-lookup service (as in the case of Lime(Picco et al., 1999)). However, the messages to be matched depend on the implementation, which tells why these architectures cannot describe dynamic changes of federation structures independently from the implementation defining the behavior of each smart object. Our approach is focused on the federation of smart objects through their interfaces. Each smart object has a set of ports corresponding to service providing ports or to service requesting ports. Thus the federation of the smart objects does not depend on their implementation, nor on a repository-and-lookup service. This means that, since our approach does not require any repository, we have a new

Julia J., Tanaka Y. and Spyratos N..
FORMALIZATION OF AN RNA-INSPIRED MIDDLEWARE FOR COMPLEX SMART OBJECT FEDERATION SCENARIOS.
DOI: 10.5220/0003818400960105
In *Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems* (PECCS-2012), pages 96-105
ISBN: 978-989-8565-00-6
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

federation model for purely P2P networks.

The formal models that we aim to provide deal with the following three different levels:

- First Level: The port matching model is used to describe federation and interoperation mechanisms.
- Second Level: Graph rewriting rules are used to describe dynamic change of federation structures.
- Third Level: The catalytic reaction network modeling is used to describe complex application scenarios with mutually related more than one federation.

A catalytic network reaction is a network of catalytic reactions in which a product of a reaction may work as a source material for another reaction or as a catalyst to enable (stimulus) or inhibit (inhibitor) another reaction. In our research the materials are smart objects. To implement this system, we use what we call nucleotide smart objects. The nucleotide smart object framework is described at the second layer of our formal model, i.e.; it describes the dynamic change of federation structures. In this framework, smart objects have behaviors similar to the nucleotides in the RNA world hypothesis. By simulating the replication of RNA, we can describe all kinds of catalytic reaction networks that we are interested in (i.e., federation and unfederation with or without a stimulus as a catalyst). This paper is focusing on the mapping from the third level to the second level. In the next session, we will define what a catalytic reaction network denotes. In the following section, we will shortly review the definition of smart objects. After that, we will show what nucleotide smart objects denotes and how we can use them to model each reaction in catalytic reaction networks. Then, we will present the global state transition diagram that summarizes all the rules of the nucleotide smart objects and helps us to explain the whole process of implementing each catalytic reaction. Finally, we will give an example application of this new framework.

2 CATALYTIC REACTION NETWORK

Stuart A. Kauffman believes that a collectively autocatalytic set is one of the essential mechanisms for the self-organization of life (Kauffman, 2000). We believe that complex application scenarios of autonomic federation of smart objects can be modeled as a catalytic reaction network. A catalytic reaction network is a set of reactions in which the product of each reaction may work as a source material of another reaction or as a catalyst to enhance or to repress another reaction. Each reaction is either a composition to produce a compound product from more primitive source materials, or a decomposition of a source material into its component materials.

In our modeling, the materials are smart objects. For example, Fig. 1 shows four reactions (R1, R2, R3 and R4).

All characters outside a circle represent a smart object or a composite smart object. A reaction between them can happen with the influence of a stimulus (for example the reaction R3 between PH and T to produce PHT will happen only with the presence of MI) or without the influence of a stimulus (for example the reaction R1 between M and I to produce MI), and with or without a specific context that is made of smart objects and represented by an object in each reaction circle. The result material of a reaction may work as an input material (for example PH), or as a stimulus (for example MI) of another reaction. Both stimuli and contexts work as catalyst. Stimuli are mobile, while contexts are immobile.

This application allows an automatic federation of wireless headphones and PDA depending on the users mobility in a museum. A user with a social ID card I and a smart member card M passes a check-in gate G1 of an museum. This gate sets up a federation between M and I to check if he is a registered member of this museum. The user could pick up at the entrance of the museum a headphone and a PDA. When he will pass through a second gate G2 working as a context, his PDA and headphone will be automatically connected. Later when he will come close to a table T, the federation MI works as a stimulus which may also contain his preferences. The federated objects PDA-headphone could become connected with the table to retrieve the related information. The user will receive sound and text contents in his favorite language and/or in his favorite color contrast for the text. If the user does not have a member card, he should have a ticket u. If a user come close to a table T, through the influence of u, his federated PDA-headphone object will not be connected with T, but with T' that provides generic information about the table T for non member of the museum. No user operation is necessary to set up the necessary connections of this scenarios.

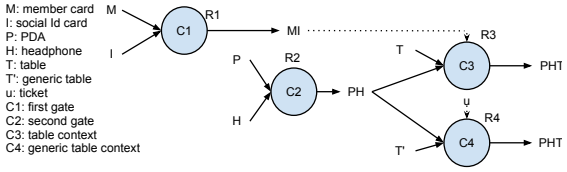

Figure 1: Catalytic reaction network representing a scenario of smart-object Federations.

In addition, we have two kinds of reactions, composition and decomposition reactions respectively corresponding to federating and defederating actions. In this paper, we will focus on composition reactions with a context and with or without a stimulus.

3 SMART OBJECT

In this section, we shortly review the definition of smart objects (SOs).

3.1 Ports

Each port has:

- a type;
- a polarity, + or − (the + polarity means that the port is a service-providing port; the − polarity means that the port is a service-requesting port);
- an arity (a service-requesting port has an arity of one; a service-providing port can have an arity greater than one);
- a state, a port can be in one of the two states: Visible (V) or Hidden (H);

The type and polarity are permanent characteristics of a port, whereas the state of a port can change.
A SO can have many ports but no two of them can have the same type and the same polarity (but they can have the same type if they have different polarities).

3.2 Connections

Here are described the characteristics of a connection from a SO A to a SO B:

- There can be a connection from A to B only if A has a requesting port and B has a providing port of the same port type. The common port is defined to be the type of the connection.
- There can be at most one outgoing connection from a given port of a smart object.
- There can be any number of incoming connections to a given port of a smart object.
- There can be many connections from A to B using different ports.

3.3 Smart Object Graph (SOG)

A smart object graph is an edge-labeled directed graph without self loops, in which:

- The nodes are smart objects;
- There is an edge labeled t from a node A to node B if there is a connection of type t from A to B.
- Each node is associated with the following five properties:
  - the smart object identifier denoted oid;
  - the smart object type;
  - the smart object state;
  - the set of its visible ports;
  - the set of its hidden ports;

  The set of these properties is called the description of the node. Note that, in a SOG, a path σ from a source SO A to a target SO B is uniquely defined by the sequence of the labels of its edges. This follows from the fact that a requesting port can have at most one connection.

A SOG is accompanied by another graph called the proximity graph (PG) defined as follows:

- the nodes are those of SOG;
- there is an edge from A to B if B is in the scope of A;

where the scope of A is defined to be the set of smart object whose existence can be detected by A, say, through wireless communication.

3.4 Graph Rewriting Rules

Graph rewriting rules are the second-level formal modeling of proximity-based federation that was introduced by (Tanaka, 2010). It focuses on the dynamic changes of interconnection structures among smart objects in a single complex federation. This model describes a rule as two SOGs. Both of them are sharing the same nodes. The first one, called the condition part, represents the condition of the application of the rule. The second, called the action part, corresponds to the change result of the condition part after applying some primitive actions. The case where the rule can be applied is illustrated by showing the condition on the left hand side. The result of the rule application is specified by the SOG on the right hand side. The condition part SOG can be extended with edges from its proximity graph.

Primitive Conditions. Each primitive condition is presented in Fig. 2. The gray node means the smart object that checks each condition and executes the rule. It is called the activated node. For each condition, the path σ may be of length zero. A proximity graph edge is represented by a dotted arrow.

Port Condition: the σ object has a certain port p in a certain state. The letter "v" is used for the visible state, and the letter "h" for the hidden state.

Port Connections: the σ object has a port with a certain number of active connections (0 is the example). Note: the notation -p:v(1) denotes that a port -p is visible with 1 active connection.

Type Condition: the σ object is of type t.

State Condition: the σ object has the state S.

Scope Condition: the σ2 object is in the scope of the σ1 object.

We defined a list of commands that will be used to check the previous conditions and allow to get the oid, the type, the state, the state of one of the ports, the number of active connections of one of the ports, the presence of a σ2 object in the scope, and the presence of an object with a certain port in the state visible in the scope of a σ object.

We defined a list of commands that will be used to check the previous conditions and allow us to get for a σ object its oid, its type, its state, the state of one of its ports, the number of active connections of one of its ports, the presence of a σ2 object in its scope, and the presence of an object with a certain port in the state visible in its scope.

Primitive Actions. Each possible action is shown in Fig. 3.

Port Exposition: the activated node makes the port -p of the σ object visible.

Port Hiding: the activated node makes the port -p of the σ object hidden.

State Setting: the activated node sets the state of the σ object to S.

Connection Breaking: remove the edge labelled p from the σ object.

Channel Spanning: span an edge labelled p from the σ1 object to the σ2 object.

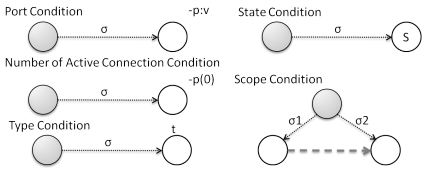

Figure 2: Primitive conditions that can be used in graph rewriting rules.

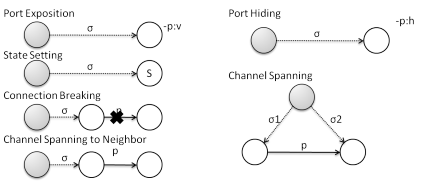

Figure 3: Primitive actions that can be used in graph rewriting rules.

Channel Spanning to Neighbor: span an edge labelled p from the σ object to an object with a -p port.

We defined a list of commands that will be used to execute the previous actions and allow us to get for a σ expose of hide one of its ports, to set its state, to span or to break a connection to a σ2 object or to an object in its scope.

Complex Rules. As shown in Fig. 9, it is possible to combine the different primitive conditions and actions to define such a complex rule as the rule 2. For $\sigma 1 = nil$, and $\sigma 2 = B[x]$, this rule check that the σ1 object and the σ2 object have the state a and 0, and that the σ1 object is in the scope of the σ2 object. The actions consists to set the state of the σ1 object and the σ2 object to c and 2, and to span a connection from the σ2 object to the σ1 object through the port $B[x]$.

4 NUCLEOTIDE SMART OBJECTS AS ELEMENTS OF CATALYTIC REACTION NETWORKS

4.1 Nucleotide Smart Objects

Nucleotide smart objects (NSOs) are a solution to implement the primitives of the third layer: the catalytic reaction network. The idea is to emulate the RNA replication process in the smart object world to design reactions of a catalytic reaction network in a generic way.

We use different types of NSOs corresponding to different types of SOs. For example, PDAs, credit cards, or earphones are different types of SOs. We use a different type of NSOs for each different type of them. A NSO of subtype t is a SO in which there are exactly four ports:

- two ports with the same type L; one of them is service-requesting and the other is service-providing, hence denoted as -L and +L respectively;
- two ports with the same type; the type of these ports is dependent of the subtype of the NSO. It is denoted as B[t] where t is the subtype of the NSO; one of them is service-requesting and the other is service-providing, hence denoted as -B[t] and +B[t] respectively;
- each of these ports can have at most one connection at any given moment;

Strand of Nucleotide Smart Objects. Before defining the inputs of a reaction, we define a strand as an SOG, and some operations on a strand. A strand is a maximal acyclic path such that:

1. all nodes are NSOs;
2. all edges have the same label L;
3. no two nodes in the path are connected by a B edge.

Given a strand $V : v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n$, a complement of V is defined to be any strand $\bar{V}$ such that:

1. $\bar{V} : u_1 \rightarrow u_2 \cdots \rightarrow u_n$
2. $type(u_i) = type(v_{n-i+1}),\ i = 1, \cdots, n$

4.2 Inputs Strands of a Reaction

The input materials of a reaction may be a set of k input strands, $V_1 \cdots V_k$; there is at most one distinguished input strand that plays the role of a stimulus. We assume (without lost of generality) that, if a stimulus is involved, the strand V_1 is the stimulus. We also assume that each NSO of subtype t used in input strands is already federated with a SO of type t. For example in Fig. 4, you have an example of the input materials of the context C3 presented in Fig. 1.

4.3 Context of a Reaction

Another element of a reaction is a strand called context defined as follows:
$\bar{V}_1 \rightarrow S_1 \rightarrow \bar{V}_2 \rightarrow S_2 \rightarrow \cdots \rightarrow \bar{V}_{k-1} \rightarrow S_{k-1} \rightarrow \bar{V}k$
where $type(S_i) = I$ (Input separator) $, i = 1, \cdots, k-1$ and if V_1 is a stimulus, then $type(S_1) = S$ (Stimulus separator).
Later, each $\bar{V}_i$ of a context is called a docking part.

For example in Fig. 5 is presented the implementation of the context C3 presented in Fig. 1.

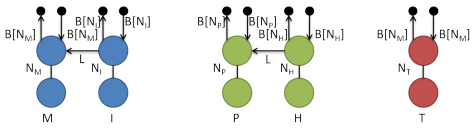

Figure 4: Input materials of the reaction R3 in Fig. 1 using C3 and MI as its context and stimulus.

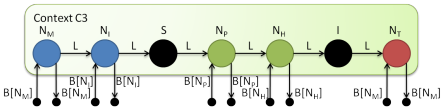

Figure 5: The context C3 of the reaction R3 in Fig. 1.

4.4 Output of a Reaction

If V_1 is a stimulus, then the output is $V_2 \rightarrow V_3 \rightarrow \cdots \rightarrow V_n$; else, the output is $V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_n$.

In Fig. 8 is presented the output of the reaction that happens in the context C3.

4.5 Implementation of a Reaction

A context strand will work as a template defining how to federate the objects. For example C3 (Fig. 5) is a context of a reaction accepting as input a stimulus MI and two input materials PH and T. When the corresponding SOs M, I, P, H and T attached to their NSOs (Fig. 4) enter in the scope of the context C3, each NSO of the input strands become automatically connected to the corresponding docking part of the context (Fig. 6). After this step, the connection among the NSOs in the context through the ports L is replicated in the opposite direction among the NSOs in the stimulus and in the input materials (Fig. 7).

There, we first need to break all the connections between the context strand and both the stimulus and the input materials, and then need to break the connection between the stimulus and the object P. As a result, we have the federation PHT (Fig. 8).

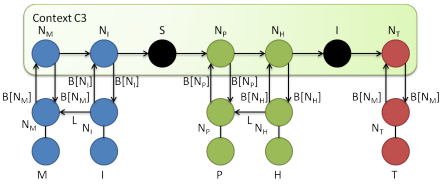

Figure 6: The docking of the stimulus strand and two input strands.

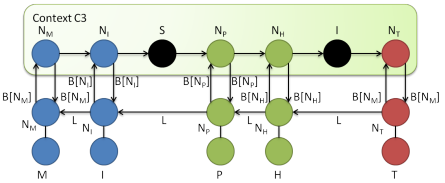

Figure 7: Composition of an output strand from two input strands.

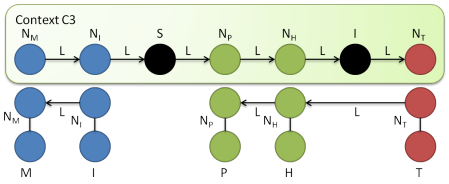

Figure 8: The undocking ot the stimulus strand and the output strand.

Because T is indirectly connected to H and P through the NSOs, some software that had been stored in the context can be sent to T to establish arbitrary direct connections among the three SOs P, H and T.

Suppose that the object P can be any kind of these PDAs. We can consider for example a Microsoft Windows Mobile PDA, and a Palm OS PDA. We can develop a NSO, for the Microsoft Windows Mobile PDA and another for the Palm OS PDA, having the same subtype "N_{PDA}", and will communicate with other NSOs with the same protocol. We will be able to use all these different PDAs in the same federation context without redesigning them.

All the reactions designed in the same way share the same implementation structure. This structure is composed of:

- the first layer, that is called the first strand or the context;
- the second layer, that is called the second strand, corresponding to the set of input strands;
- the third layer, corresponding to the SOs, working as a stimulus or as input materials, to be combined.

The rest of the paper will focus on the first and second layers of the structure, i.e., the first strand and the second strand.

4.6 Graph Rewriting Rules for Nucleotides Smart ObjectS

We assume that the context is manually constructed as follows.

- The first node of the context has the state 0 and the ports $+B[N_i] : v(0)$, $-B[N_i] : h(0)$, $+L : h(0)$ and $-L : h(1)$.
- The last node of the context has the state: 1 and the ports $+B[N_i] : h(0)$, $-B[N_i] : h(0)$, $+L : h(1)$ and $-L : h(0)$.
- Each intermediate node of the context has the state 1 and the ports $+B[N_i] : h(0)$, $-B[N_i] : h(0)$, $+L : h(1)$ and $-L : h(1)$.

The context with the above mentioned nodes is said to be in the initial state. As we shall see, after obtaining the output, the context is automatically reset to its initial state.

In each input strand, if a strand contains a single node, then the description of this node includes the followings:

- the state a and the ports $+B[N_i] : h(0)$, $-B[N_i] : v(0)$, $+L : h(0)$ and $-L : h(0)$;

else, the description of these nodes includes the followings:

- The first node of the input strand has the state b and the ports $+B[N_i] : h(0)$, $-B[N_i] : v(0)$, $+L : h(1)$ and $-L : h(0)$.
- The last node of the input strand has the state a and the ports $+B[N_i] : h(0)$, $-B[N_i] : h(0)$, $+L : h(0)$ and $-L : h(1)$.
- The last node of the input strand has the state 1 and the ports $+B[N_i] : h(0)$, $-B[N_i] : h(0)$, $+L : h(1)$ and $-L : h(1)$.

We will make an assumption that the connections are stable and not broken unless explicitly broken. Furthermore, we will assume that we will get only the expected input strand for each docking part. If not, and if the input strand has already started to dock with the docking part, we should undock the input strand, and try to find another one. The rules managing unexpected input strands, however, will not be given in this paper because of the page limitation.

In the Fig. 9 are presented the graph rewriting rules of the NSOs. For the rules 1 to 15, when the type of a SO is not specified, the type is N. In the rules 16 to 22, a smart object can be of type N, S or I. By following these rules, step by step, the input strings will dock at the docking part of the context from left to right, and then the different input strings will horizontally be connected together. The NSOs are undocked from the context from right to left. Then we get a new federation of NSOs corresponding to the federation of the different input strings. This new federation is called the output material of the reaction and it can be used as an input string of another reaction.

5 CORRECTNESS OF THE GRAPH REWRITING RULES FOR NSOS

It is difficult to visualize and to verify the full process that is obtained from these rules. To solve this problem, here, we will introduce a Global State Transition Diagram (GSTD) to summarize the rule execution process. The first part of this section presents the graph rewriting rules as mathematical rules and how to generate the global state transition diagram corresponding to these mathematical rules. The second part of this section is a description of the GSTD that is obtained from the NSO graph rewriting rules. And in the third part, we use this GSTD to explain the whole process of a reaction.

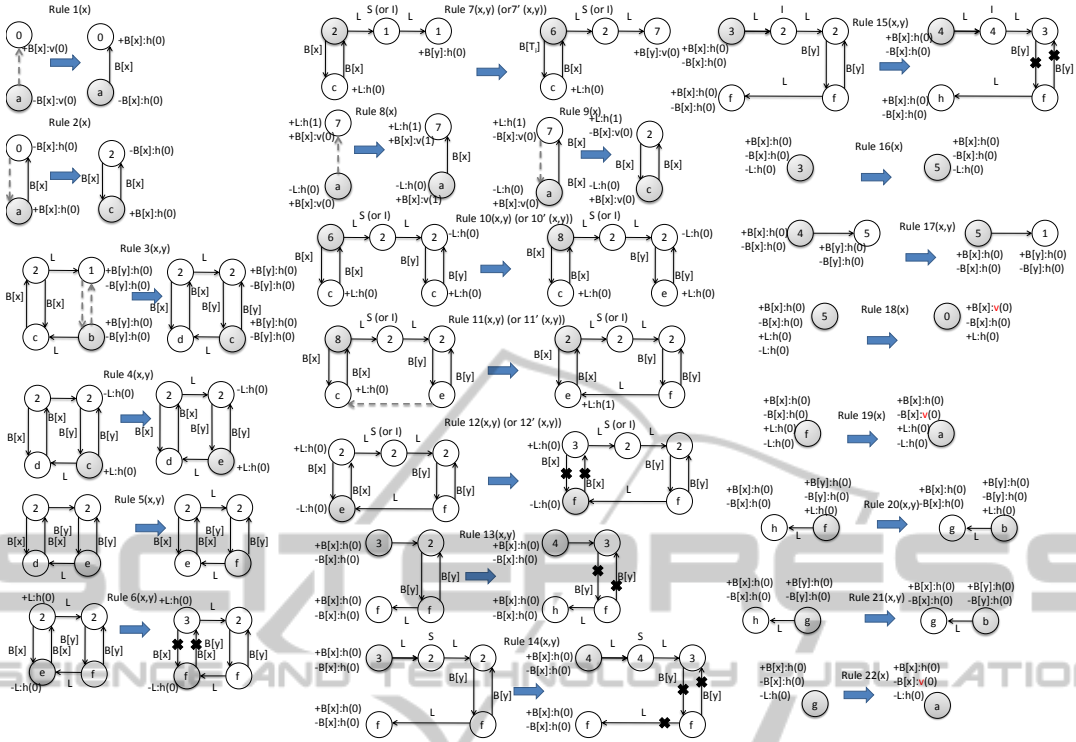

Figure 9: Graph rewriting rules for nucleotides smart object.

5.1 Graph Rewriting Rules and their Dependencies

5.1.1 Precondition and Postcondition

The precondition of a rule is the SOG in the condition part of the rule, where we neglect all edges from the proximity graph. The postcondition of a rule is the SOG in the action part of the rule.

We define the following functions:
$PreCd(R)$: to return the precondition of the rule R;
$PostCd(R)$: to return the postcondition of the rule R.

5.1.2 Injective SOG Homomorphism

An injective SOG homomorphism (ISOGH) from an SOG X to an SOG Y is an injection f from the nodes of X to the nodes of Y such that:

- For every node n of X, n and $f(n)$ have the same type and the same state.
- For every node n of X, if both n and $f(n)$ have ports of the same port type and of the same polarity, then these ports must have the same state and the same number of active connections.
- For every edge in X labelled t from a node n_1 to a node n_2, there exists in Y an edge labelled t from $f(n_1)$ to $f(n_2)$.

X is embedded in Y (denoted $X \prec Y$) if and only if there exists an ISOGH from X to Y.

5.1.3 Joinability between Two SOGs

It is possible to join two SOGs X and Y through an SOG Z if and only if it exists an ISOGH f_x from Z to X and another ISOGH f_y from Z to Y such that:

- for every edge labelled t of X from a node n, is there exists a node m in Z such that $f_x(m) = n$, them there is an edge with the same label t in Z from the node m, or there is no such an edge in Y from $f_y(m)$;
- for every edge labelled t of X to a node n, is there exists a node m in Z such that $f_x(m) = n$, them there is an edge with the same label t in Z to the node m, or there is no such an edge in Y to $f_y(m)$;

A join consists of replacing the nodes of X corresponding to the nodes of Z by their respective corresponding nodes in Y to obtain a SOG W, as shown in Fig. 10. The joinability function takes as its inputs three SOGs X, Y and Z, and returns 1 as its output if the two SOGs X and Y can be joined through a common SOG Z, or 0 otherwise.

In the example shown in Fig. 10 f_{x_1} and f_{x_2} are two ISOGHs from Z to X, f_{y_1} is one ISOGH from Z to Y and the node ids are written in the black box.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

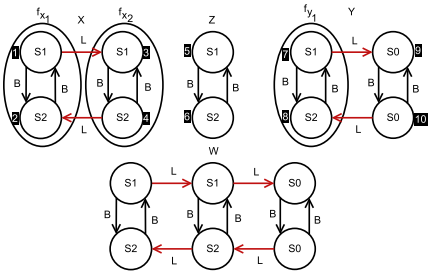

Figure 10: Example joinability between two SOGs.

For example, let us consider the two SOG homomorphisms f_{x_1} and f_{y_1} in Fig. 10. It is not possible to join X and Y through Z because there is an edge labelled L in X from the node $f_{x_1}(5)$ (corresponding to the node 1), but there is no edge with the same label L in Z from the node 5, and furthermore, there is an edge with the same label L in Y from $f_{y_1}(n)$ (corresponding to the node 7).

However, if we consider the two ISOGHs f_{x_2} and f_{y_1}, it is possible to join the two SOGs X and Y, because,

- for every edge labelled t of X from a node $f_{x_2}(n)$, either there is an edge with the same label t in Z from the node n, or there is no edge with the same label t in Y from $f_{y_1}(n)$, and
- for every edge labelled t of X to a node $f_{x_2}(n)$, either there is an edge with the same label t in Z to the node n, or there is no edge with the same label t in Y to f_{y_1} .

5.1.4 Modified Part of a Postcondition

The modified part of a postcondition of a rule R is a subgraph of this postcondition that is obtained by removing all the edges that keep the same description and/or keep the same adgency nodes with the same edges, between the precondition of R and the postcondition of R.

We define a function $ModPostCd(R)$ that returns the modified part of the postcondition of the rule R as an SOG. For example, in Fig. 11 $ModPostCd(R)$ is the SOG within an oval.

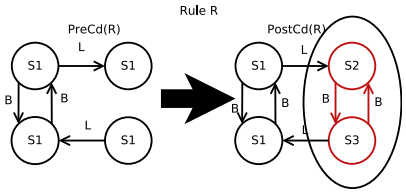

Figure 11: The modified part of the postcondition of an example rule.

5.1.5 Rule Dependency

Let $SOGs$ be the set of all the possible SOGs.

The rule dependency (RD) is a directed graph $D_D = (V_D, E_D)$ where the set V_D of vertices is the set of rules and the set E_D of edges is defined as the following set:

$$E_D = \{(R,S) | (R,S) \in V_D^2 \wedge \exists F \in SOGs\ (\exists G \in SOGs\ (G \prec F \wedge G \prec ModPostCd(R)) \wedge joinability(PostCd(R), PreCd(S), F) = 1))\} \quad (1)$$

An edge from a rule R to a rule S denotes that the execution of the rule R may lead to the triggering of the rule S. the execution of a rule R may trigger the execution of a rule S if and only if it is possible to join $PostCd(R)$ and $PreCd(S)$ through an SOG F, and if this SOG F contains at least one node of $ModPostCd(R)$.

5.1.6 Extended Rule Dependency

The extended rule dependency (ERD) is a directed graph where the set of vertices is the set of preconditions and postconditions of the rules. It has an solid edge from the precondition to the postcondition of the same rule. It has also an dotted edge from a postcondition of one rule to a precondition of another rule if there exists an edge in the rule dependency from this postcondition this precondition.

5.2 Global State Transition Diagram

The GSTD is defined based on the simplified ERD diagram. The GSTD shown in Fig.12 summarizes only the rule 1 to 15 presented in Section 4.6. Each vertex of this diagram represents an SOG (either a precondition, or a postcondition). Here, types and ports are not specified. In each vertex, all the horizontal connections are made through ports -L and +L, and all the vertical connections through ports $-B[x]$ and $+B[x]$. We will use capital letters associated with each vertex to refer to its SOG. Each solid edge between vertices represents an execution of a rule, and is called a "transition". Each number used as a label to a transition corresponds to the number of the rule to be executed. Each dotted edge from a postcondition of a rule R to a precondition of a rule S denotes that, after the execution of the rule R, an SOG corresponding to the precondition of the rule S may be obtained and the rule S may be executed. For example, in Fig. 13, by making the transition C to D, the SOG D is obtained. At the same time, D also creates I. Thus, dotted edges are called "transition dependencies". Each transition

SCIENCE AND TECHNOLOGY PUBLICATIONS

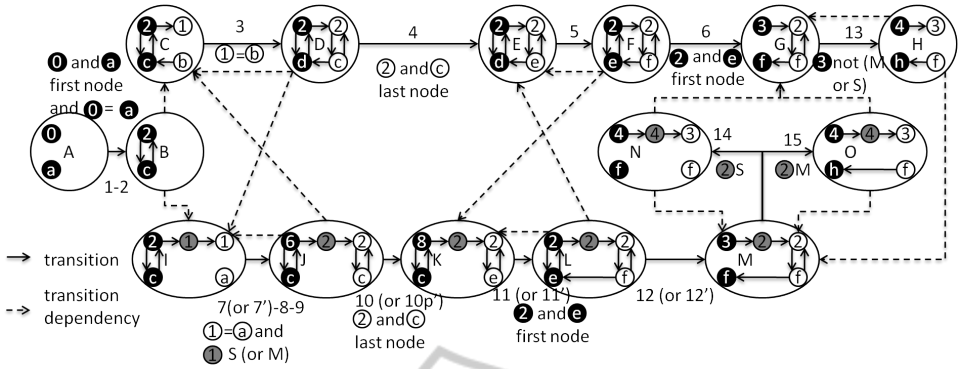

Figure 12: A part of the global state transition diagram for the graph rewriting rules in Fig. 9.

is identified by the concatenation of the source SOG name and the destination SOG name. For example, the transition from C to D is called the transition CD.

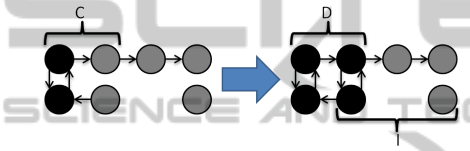

Figure 13: Transition dependency.

When in the RD diagram we have a sequence of rules $(R_1, R_2, ...R_n)$ such that each rule (except R_n) has only one outgoing edge to its following in the sequence, and each rule (except R_1) has only one incoming edge from its predecessor in the sequence, we define in the GSTD only the precondition of the first rule of this sequence, and the postcondition of the last rule of this sequence. For example, from the RD diagram, we could see that after the execution of the rule 7 (or the rule 7'), only the rule 8 can be executed, and after the rule 8, only the rule 9 can be executed. Thus, the transition IJ corresponds to the execution of the sequence of rules 7 (or 7'), 8 and 9.

When there exists a dotted edge from a $postCd(R)$ to a $preCd(S)$, and if $postCd(R) \prec preCd(S)$ and $preCd(S) \prec postCd(R)$, we may merge the postcondition of the rule R with the precondition of the rule S. For example, $postCd(3) \prec preCd(4)$ and $preCd(4) \prec postCd(3)$. The only difference is that there are two primitive conditions in 4 (two ports of type L should be in their hidden state with 0 active connection, see Fig. 9), that are not in 3. Thus, the transition CD represents the execution of the rule 3, and the transition DE represents the execution of the rule 4. D is the postcondition of the rule 3 and also the precondition of the rule 4. We may add the two extra primitive conditions to the transition DE, the white node in state 2 in the SOG D should be the last nodes of the first strand and the white node in state c in the SOG D should be the last node of the second strand. Because the ports and the types are not specified in the GSTD, the port and type conditions may be expressed as extra conditions on the transitions. These extra conditions are directly inherited from the graph rewriting rules. There are three different types of basic conditions that are used and combined to specify each transition conditions:

Type Conditions: They check the type of a node. For example in the transition IJ, the condition "gray node 1 M" means that the gray node in state 1 should be of type M (input Material separator);

Subtype Condition: They compare the subtypes of two NSOs. For example in the transition IJ, the condition "white node 1 = white node a" means that the white node in state 1 should have the same subtype as the white node in state a;

Position Conditions: They check the position of a node in the context. For example, in the transition AB the condition "black node 1 and black node a first node". There are only four positions to be specified. The first node of the context strand (a node with a +L:0 port), the first node of an input strand (a node with a -L:0 port), the last node of the context strand (a node with a -L:0 port) and the last node of a input strand (a node with a +L:0 port).

5.3 Outline of the Correctness Proof of our NSO Rules

The whole process is divided into four stages: docking, output strand formation, undocking and reinitialization. Each stage consists of "propagation of SOGs" over the global structure. Basically, for each transition, other SOGs reached through dotted arrows may be also obtained as explained previously. This corresponds to the "SOG propagation".

Docking: This stage connects different input strands to the context strand.

Initialization: By the transition AB, we get either

C or I.

Propagation: From C (, or I), it is only possible to perform CD (, or IJ). Both of these transitions connect a node of the context to a node of an input strand . After the transition CD (, or IJ), if the white node in state 2 is not the last node of the first strand, we will obtain either C or I, and, hence, we will repeat the same step. Else we can only obtain the transition DE (, or JK), to obtain a E (, or K).

Input Strand Interconnection: When the last node of the last input strand is connected to the last node of the context, all the input strands are correctly docked (because the direction of the docking is from the left to the right, and the last node of the context is its right most node).

Initialization: By the transition DE (, or JK), we will get E (, or K).

Propagation: From E (, or K), it is only possible to perform the transition EF (, or KL). Both of these transitions change the state of the black node to the state d (JK also connect the white node in e to the black node in c). After the transition EF (, or KL), if the black node in 2 is not the first node of the first strand, we will obtain either E or F, and we can only repeat the same step. Else we can only perform the transition FG (, or LM), and we will obtain G (, or M).

Undocking: This stage undocks the output strand and the stimulus by breaking the connections between the nodes in the first strand and the nodes in the second strand from right to the left.

Initialization: By the transition FG (, or LM), we will get G (, or M).

Propagation: From G, we can only perform the transition GH (if the black node in 3 is not a separator). From M, if the grey node in 2 is a stimulus separator or an input separator, we can perform either the transition MN or MO. Therefore we can perform either the transition GH, MN or MO. If the white node in 3 is not the last node of the first strand, we will obtain either G or M, and will repeat the same step.

Reinitialization: When the undocking stage is terminated, we have a pair of a context and a output strand, or a triple of a context, an output strand and a stimulus. They must be reinitialized to be able to participate in another reaction. This is done by the rules from 16 to 18 for the context and by the rules from 19 to 22 for the stimulus and the output strand.

6 CONCLUSIONS

This paper has proposed a framework based on the RNA replication mechanism to design the third level of our formal modeling. With this framework, it is possible to implement any reactions of a catalytic reaction network by using nucleotide smart objects in a generic way. In order to prove the validity of such an implementation, we have introduced a global state transition diagram that resumes the graph rewriting rules. The NSOs in the same strand need not to be in the same place. They may exist at different locations in a physical environment and be connected through the Internet. This modeling opens a new vista for new application scenarios of proximity-based federation of smart objects. With this new approach based on the RNA replication mechanism, and with the representation of the interactions among smart objects based on catalytic reaction networks, we will become able to consider, to describe, and to implement new innovative scenarios beyond the current scope of stereotyped applications of ubiquitous, pervasive, and/or mobile computing.

REFERENCES

Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112.

Henricksen, K., Indulska, J., and Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. *in F. Mattern and M. Naghshineh (Eds.): Pervasive 2002, LNCS*, 2414:167–180.

Kauffman, S. A. (2000). *Investigations*. Oxford University Press.

Milner, R. (2001). Bigraphical reactive systems: Basic theory. In *Proceedings of the International Congress of Mathematicians*, pages 155–169.

Milner, R. (2004). Theories for the global ubiquitous computer. *Foundations of Software Science and Computation Structures, LNCS*, 2987:5–11.

Picco, G. P., Murphy, A. L., and Roman, G.-C. (1999). LIME: Linda Meets Mobility. In *Proc. of ICSE'99 Int'l Conference*, pages 368–377.

Tanaka, Y. (2008). Proximity-based ad hoc federation among smart objects and its applications. *Proc. of CSTST'08 int'l.*

Tanaka, Y. (2010). Proximity-based federation of smart objects: Liberating ubiquitous computing from stereotyped application scenarios. *Proc. of KES'10 Int'l Conference, LNCS*, 6276/2010:14–30.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS