A Formal Definition of RESTful Semantic Web Services

Antonio Garrote Hernández
University of Salamanca
Salamanca, Spain
agarrote@usal.es

María N.Moreno García
University of Salamanca
Salamanca, Spain
mmg@usal.es

ABSTRACT

In this article a formal model applying REST architectural principles to the description of semantic web services is introduced, including the discussion of its syntax and operational semantics. RESTful semantic resources are described using the concept of tuple spaces being manipulated by HTTP methods that are related to classical tuple space operations. On the other hand, RESTful resources creation, destruction and other dynamic aspects of distributed HTTP computations involving coordination between HTTP agents and services are modeled using process calculus style named channels and message passing mechanisms.
The resulting model allows for a complete and rigorous description of resource based web systems, where agents taking part in a computation publish data encoded according to semantic standards through public triple repositories identified by well known URIs. The model can be used to describe complex interaction scenarios where coordination and composition of resources are required. One of such scenarios taken from the literature about web services choreography is analyzed from the point of view of the proposed model. Finally, possible extensions to the formalism, such as the inclusion of a description logics based type system associated to the semantic resources or possible extensions to HTTP operations are briefly explored.

1. INTRODUCTION

In recent years, semantic web services [1] and RESTful [2] web services have been two important topics for researchers and practitioners in the field of distributed web computation, although they have remained largely unrelated.
RESTful web services have achieved a great success in their application to actual web development but limited research have been undertaken in their formalization and expressiveness. On the other hand, ongoing effort in the standardization of semantic web services technology draws heavy influence from WS-* standards and retains strong RPC semantics. Incipient proposals for developing more RESTful semantic web services like hRESTS [3] or SA-REST [4] are working to bring the world of semantic web services closer to the field of RESTful services. Nevertheless, different foundations for the development of semantic web services like triple space computing models have also been proposed [5]. This model shows remarkable similarities with the architectural principles of HTTP and REST. This model can be used to describe a kind of RESTful semantic web services characterized by the following points:

- A RESTful semantic resource consists of a set of triples stored in a certain shared memory repository accessible by processes taking part in a computation.
- The triple space can be accessed through an associated URI that can be linked from related resources.
- The triple space containing the resource's triples can be manipulated using HTTP requests issued to the resource URI according to REST semantics.

This kind of RESTful web services may be applied to a wide range of actual web resources from a XHTML page with embedded RDFa [6] triples, to a web application connected to a triple store. It is also conceptually similar to the semantic web standard for the SPARQL protocol [sparqlprot].

In this paper a formal calculus for distributed computation using this kind of RESTful semantic web services will be described. This formalism merges aspects from triple space computation and other formalisms like process calculi [7] to build a generic abstraction of the computational model underlying RESTful semantic web services distributed computation.

2. SEMANTIC RESOURCES AND TRIPLE SPACES

The basis of the calculus is the manipulation of semantic meta data represented as triples. Each triple has subject, predicate and object, as described by the Resource Description Framework W3C's recommendation [10]. Components of a triple (v) can consist of URIs (μ) or literals (λ). Every computation described in the calculus consists in the manipulation of triples stored in shared data spaces known as triple spaces [5] (θ_i) by a set of distributed processes $(P, Q, ...)$.
This model of distributed computation known as *generative communication* [11] was pioneered by the Linda system for distributed computation. Linda original operations are

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WS-REST 2010, April 26, 2010; Raleigh, NC, USA
Copyright 2010 ACM 978-1-60558-959-6/10/04 ...$10.00.

$$
\begin{array}{lcl}
P & ::= & 0 \mid T \mid P|P \mid !P \mid if\, T\, ?\, P.P \mid x ::= T \\
T & ::= & rd(\theta_i, p) \mid in(\theta_i, p) \mid out(\theta_i, v) \mid swap(\theta_i, p, v) \mid \\
 & & rdb(\theta_i, p) \mid inb(\theta_i, p) \mid notify(\theta_i, \rho, v) \\
 & & \\
\theta & ::= & \{ \text{ triple spaces } \} \\
\rho & ::= & \{in, out\} \\
\mu & ::= & \{\text{URIs}\} \\
\lambda & ::= & \{\text{literals}\} \\
p & ::= & \{\text{patterns}\} \\
v & ::= & \{\text{values}\} = \{\mu\} \cup \{\lambda\} \cup < p, v > \cup < p, \theta_i >
\end{array}
$$

Table 1: triple space operations syntax

used to describe the different ways in which a triple space can be manipulated. Symbol rd designates an operation for reading triples from a triple space without removing them, in symbol for reading and removing the triples , and out symbol defines an operation for inserting new triples in the triple space. Blocking versions of rd and in operations are also defined. In addition to these operations, the $notify$ [12] operation, which allows processes to be notified when other process manipulates the triple space, and the atomic $swap$ operation [13], combining read and write in a single operation, are also defined.

A process in the calculus is defined as a finite sequence of operations over the triple space. Parallel composition of processes, replication and a simple type of conditional based on the number of triples recovered from a triple by an operation space and simple name matching of values are defined.

Triple space operations accept as arguments sets of triples (v) or patterns (p) that must be matched against triples in the triple space. No particular pattern mechanism is chosen in this definition of the calculus. It can be assumed that patterns are a built using a subset of the SPARQL [14] query language with basic name substitution [16]. A pattern can be matched against a collection of tuples ($< p, v >$) or a tuple space ($< p, \theta_i >$) obtaining as a result a collection of bindings for the pattern variables. Each of these bindings, together with the original pattern, can be transformed into a set of matched tuples or the empty set.

Operational semantics for this formal notation follows previous formalizations of Linda type systems [11]. We have chosen to define *ordered* semantics for the triple space operations. As a consequence, emission and rendering of messages can be regarded as a single atomic operation. It can be proved that Linda systems with ordered semantics are Turing complete [17]. Table 2 shows the semantics of the calculus using labeled transitions:

The labeled transition relation $\rightarrow$ for triple space operations is the smallest one satisfying axioms (1) - (10). Rules (1) and (2) are classical rules for parallel composition and structural congruence of processes [7]. Rule (3) defines the spawning of a new process. Rules (4) through (7) define the main operations over triple spaces and how the reduction of those rules modify the triples inside the triple space they are applied to. Rules for the blocking versions of the operation have been omitted for the sake of brevity. Rules (8) and (9) define the semantics for the $notify$ operation. They show

(1)	$\frac{P \rightarrow P'}{P\|Q \rightarrow P'\|Q}$
(2)	$\frac{P \rightarrow P'}{Q \rightarrow Q'} if\, P \equiv Q \, and \, P' \equiv Q'$
(3)	$!P.Q \rightarrow Q \mid P$
(4)	$\frac{rd(\theta_i,p).P}{rd(\theta_i,p).P \xrightarrow{<p,\theta_i>} P}$
(5)	$\frac{in(\theta_i,p).P}{rd(\theta_i,p).P \xrightarrow{<p,\theta_i>} P, \theta_i = \theta_i - <p,\theta_i>}$
(6)	$\frac{out(\theta_i,v).P}{out(\theta_i,v).P \xrightarrow{\overline{v}} P, \theta_i = \theta_i \cup v}$
(7)	$\frac{swap(\theta_i,p,v).P}{swap(\theta_i,p,v).P \xrightarrow{<p,\theta_i>,\overline{v}} P, \theta_i = \theta_i - <p,\theta_i> \cup v}$
(8)	$\frac{out(\theta_i,v).Q}{notify(\theta_i,out,p).P \| out(\theta_i,v).Q \xrightarrow{\overline{v},<p,v>} P\|Q}$
(9)	$\frac{in(\theta_i,p).Q}{notify(\theta_i,in,q).P \| in(\theta_i,p).Q \xrightarrow{<p,v>,<q,<p,v>>} P\|Q}$
(10)	$\frac{if\, T\, P.Q}{if\, T\, P.Q \xrightarrow{\overline{0}} Q}, \frac{if\, T\, P.Q}{if\, T\, P.Q \xrightarrow{\overline{v}} P}$

Table 2: operational semantics for triple space operations

the relation with in and out operations and how the pattern used as an argument for the $notify$ operation is applied to the triples removed in an in operation and written in an out operation before the reduction of a $notify$ rule takes place. Finally, rule (10) defines the semantics for simple if branching construct defined in the calculus.

From this definition of operational semantics, remarkable conceptual similarity between a triple space and a HTTP resource can be observed. The syntax and semantics defined for the triple space operations could be enough basis for the description of RESTful architectures if key aspects of the triple space model are identified with different aspects of the HTTP protocol:

- A triple space can be identified with an HTTP resource.
- The identifier of the triple space can be identified with the URI associated to a HTTP resource.
- GET HTTP operations can be defined as rd operations over the triple space.
- POST HTTP operations can be defined as out operations over the triple space.
- PUT HTTP operations can be defined as $swap$ operations over the triple space.
- DELETE HTTP operations can be defined as in operations over the triple space.

$$
\begin{array}{rcl}
P & ::= & 0 \mid T \mid M \mid P|P \mid !P \mid if\ T\ ?\ P.P \mid x ::= T \mid \\
 & & new\ \mu\ in\ P \\
M & ::= & \overline{req(\mu)}[m,p,v] \mid [m,p,v]req(\mu) \mid \overline{resp(\mu)}[c,v] \mid \\
 & & [c,v]resp(\mu) \\
 & & \\
m & ::= & \{get, post, put, delete\} \\
c & ::= & \{200, 201, 404, 401\}
\end{array}
$$

Table 3: process communication syntax

3. SEMANTIC RESOURCES AND RUNTIME PROCESSES

The simple model for the description of HTTP resources introduced in the previous section presents some important shortcomings. In the original conception of the tuple space model, the shared space is supposed to be globally available and unique. On the contrary, a convenient way to model HTTP resources involves the use of as many triple spaces as resources are being modeled. As a consequence, these resources would not be static shared spaces, but they would be created after POST operations and destroyed with DELETE operations. The definition for POST and DELETE operations discussed in the previous section, where both HTTP methods were equated to *out* and *in* triple space operations, does not fit into triple space model semantics, since *out* and *in* operations can only manipulate triples but cannot create and destroy whole triple spaces. The formalism could be extended [15] introducing new operations over triple spaces as well as triples. With these extensions, triple spaces could be thought as processes in the calculus that can be spawned, receive messages from other processes and finish their execution.

Other important feature of the triple space model that have been defined is the ability to assign URIs to triple spaces. Processes can gain access to new triple spaces by reading triples containing URIs in their components. From this point of view, triple spaces can be regarded as mobile processes as described by the Pi-Calculus [7] [8]. In this kind of process calculi, processes coordinate with each other exchanging messages through named channels that can be exchanged inside the components of the messages being received or sent by processes.

The Pi-Calculus formalism is extremely useful for the description of web computation, since the concept of named channels being exchanged between processes closely resembles the exchange of web resources containing URIs between clients and servers identified by URIs.

The calculus previously introduced can be expanded to allow this new way of inter process communication as shown in table 3. The operations described are variants of the ones described in the polyadic Pi-Calculus [9]. Messages can be sent through named channels representing URIs (μ). Messages can be requests (req) or responses ($resp$) and processes can send or receive both kind of messages. Messages themselves comprise a method (m), a pattern (p) and a value (v), in the case of requests and a code (c) with a value for responses. Additionally, a construction introducing a new URI in a process and restricting its application ($new\ \mu\ in\ P$) is also defined.

$$
\begin{array}{ll}
(11) & \dfrac{P \xrightarrow{\overline{req(\mu)[m,p,v]}} P', Q \xrightarrow{[m,p,v]req(\mu)} Q'}{P|Q \rightarrow P'|Q'} \\
 & \\
(12) & \dfrac{P \xrightarrow{\overline{resp(\mu)[c,v]}} P', Q \xrightarrow{[c,v]resp(\mu)} Q'}{P|Q \rightarrow P'|Q'} \\
 & \\
(13) & \dfrac{\overline{req(\mu)[m,p,v]}.P}{\overline{req(\mu)[m,p,v]}.P \xrightarrow{*} [c,v]resp(\mu).Q} \\
 & \\
(14) & \dfrac{[m,p,v]req(\mu).P}{[m,p,v]req(\mu).P \xrightarrow{*} \overline{resp(\mu)[c,v]}.Q}
\end{array}
$$

Table 4: operational semantics for triple space operations

Operational semantics for these operations are defined in table 4. These rules also follow closely the operational semantics defined for the polyadic Pi-Calculus. Rules (11) and (12) are adaptations to the chosen syntax of the communication rule from the Pi-Calculus, showing the reaction between a process sending a message and a process receiving a message. Rules (13) and (14) impose restrictions in the order requests and responses can be exchanged. According to these axioms, if a process sends a request through an URI, it must expect a response in a certain number of reductions. In the same way, if a process receives a request through an URI it must send a response through the same URI in a certain number of reductions.

The syntax and semantics described make possible the description of HTTP resources as processes receiving messages through URIs from HTTP agent processes. These messages consist of a HTTP operation plus a triple pattern and/or semantic meta data. Resource processes can be spawned from other processes after receiving a POST request and can finished its execution after receiving a DELETE message. Responses returned from resource processes to agent processes can also contain URIs referencing other resources.

4. MODELING RESTFUL SEMANTIC SERVICES

In previous sections, two different formalisms for distributed computing have been introduced: the triple space computing model and mobile processes calculus. Both formalism offer different coordination mechanisms among processes performing a distributed computation: distributed shared memory the former and message passing the later. Both mechanisms can be regarded as variants of a more generic interprocess coordination mechanism [16]. Nevertheless, each model is specially well suited for the description of certain aspects of computations using RESTful semantic services. The triple space formalism offers an excellent description of semantic RESTful resources as static repositories of semantic data, using operations over the stored data as the main coordination mechanism among processes manipulating the stored triples. Conversely, process calculi are well suited for the description of the HTTP protocol as a message passing mechanism through named channels (URIs associated to IPs) among web agent processes and resource processes, as well as for the description of the dynamic aspects of REST-

$$
\begin{array}{rcl}
R_{REST}(\theta,\mu) & ::= & [m,v,p]req(\mu).\, if\ m = get\ ?\ R_{get}(\theta,\mu). \\
 & & if\ m = post\ ?\ R_{post}(\theta,\mu). \\
 & & if\ m = put\ ?\ R_{put}(\theta,\mu). \\
 & & if\ m = delete\ ?\ R_{delete}(\theta,\mu). \\
 & & \overline{resp(\mu)}[406,0].R_{REST}(\theta,\mu) \\
 & & \\
R_{get}(\theta,\mu) & ::= & x ::= rd(\theta,p).\overline{resp(\mu)}[200,x].R_{REST} \\
 & & \\
R_{post}(\theta,\mu) & ::= & \overline{new\ \nu\ in\ out}(\theta,<p,\nu>).!R(\theta,\nu). \\
 & & \overline{resp(\mu)}[201,<p,\nu>].R_{REST} \\
 & & \\
R_{put}(\theta,\mu) & ::= & swap(\theta,p,v).\overline{resp(\mu)}[200,v].R_{REST} \\
 & & \\
R_{delete}(\theta,\mu) & ::= & in(\theta,p_\mu).\overline{resp(\mu)}[200,0].0
\end{array}
$$

Table 5: parametric definition of a simple semantic RESTful resource

ful semantic resources such as their creation and destruction.

In this section, a description of semantic RESTful computation is defined by combining aspects of both formalisms and using the syntax and operational semantics introduced in previous sections. The main features of this model can be summarized as follows:

- Any computation is performed by distributed processes: agents and resources.
- Any process has a number of associated triple spaces that can be manipulated using triple space operations.
- A certain number of processes sharing the same triple spaces can be grouped in a *computational locus*: a server web application or a web browser application are examples of computational loci.
- Resource processes will have an associated named channel (URI) where they can receive messages from other processes.
- Agent processes do not have associated named channels but can send messages to resource processes through URIs stored in their triple spaces.
- Intra computational locus coordination is triple space based, inter computational loci coordination is message passing based
- Named channels (URI) can be exchanged via message passing and they can be stored as part of the triples inside a triple space
- Triple space handlers cannot be exchanged via message passing

Using this model, a semantic RESTful resource can be formalized as a process with an associated triple space and URI, processing remote requests according to REST semantics: $S_{REST}(\theta,\mu)$. Table 5 shows the formalization of a basic semantic RESTful resource.

This definition of a RESTful semantic web service describes a process with an associated triple space that receives messages through an assigned URI. If the message is a *get* HTTP request the process simply applies the triple pattern received to the associated triple space, and returns the matching triples. If the message is a *post* HTTP request, the process introduces a new URI and binds the subject of the triple pattern sent in the request to the new URI, generating as a result a new set of triples. These triples are written into the triple space and a new process for the resource to be created, associated to the same triple space and the new URI is spawned. The URI of the new resource process is returned in the response message for the HTTP POST operation. A *put* HTTP request containing a pattern and a set of new triples is transformed into a *swap* operation over the associated triple space, where the triples matching the received pattern are removed and replaced by the received triples. Finally, a *delete* operation means the end of the execution of the resource process as well as the deletion from the triple space of all the triples containing the resource associated URI (p_μ).

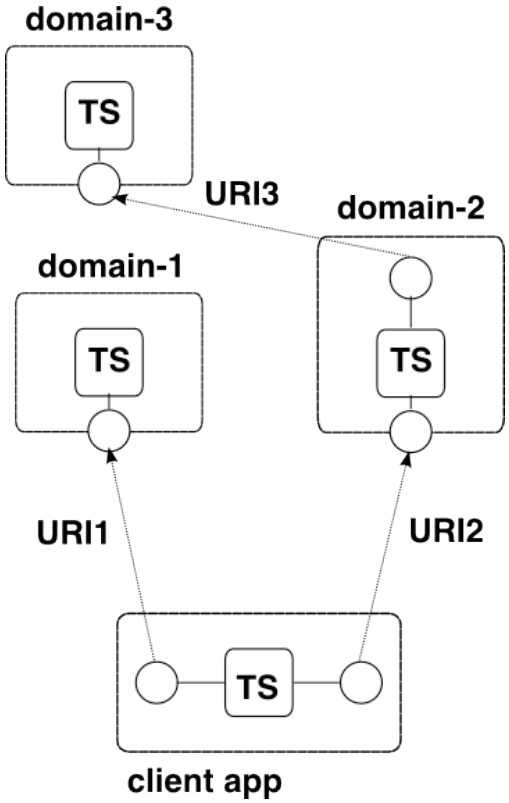

Figure 1: Sample semantic RESTful computation

Figure 1 shows a graphical representation of the proposed model displaying one client application and three different domains. Each domain hosts a semantic RESTful web service with an associated triple space and URI. The client application contains two agent processes that communicate with each other through the client application triple space. Agent processes can send messages to the resources at *domain 1* and *domain 2* through the URIs associated to each resource process. *Domain 2* also contains an agent. When

the resource at *domain 2* modifies the triple space, the agent at *domain 2* receives a $notify$ message and starts the communication with the resource at *domain 3*.

5. COORDINATION AND COMPOSITION OF RESTFUL RESOURCES

One of the main goals of the formalization of RESTful semantic services is the possibility of describing interaction patterns between resources and clients. Complex processes and workflows can be modeled as well defined and reusable descriptions, that can be automatically executed by software implementations of the calculus.

Different frameworks for web services orchestration and choreography have been proposed in the world of WS-* web services, like the W3C standard WS-CDL [18]. In this section, a simple example taken from WS-CDL literature [19], will be formalized using the calculus introduced in previous sections.

The problem discussed involves three parties: a *Buyer*, a *Seller* and a *Shipper* actors, performing a purchase transaction. The protocol is described in the following terms:

- *Buyer* asks *Seller* to offer a quote for a fixed good.
- *Seller* replies with a quote.
- *Buyer* accepts or rejects the quote.
- If buyer accepts, then *Seller* sends a confirmation to *Buyer*, then asks for delivery details to the *Shipper*.
- Finally, *Shipper* sends the delivery details to *Buyer*.

In order to describe this system using RESTful semantic web services, three different computational loci must be defined:

- The buyer client application, containing a Buyer HTTP agent process.
- The Seller web application, containing two resources: Products and PurchaseOrders.
- The Seller web application contains also a QuoteUpdater agent and a Shipper agent.
- The Buyer web application, containing one resource: ShipmentOrder.

Proper definition of the semantic meta data for the Product, PurchaseOrder and ShipmentOrder resources in some ontology definition language like OWL, as well as the precise content of the triple patterns, are not included for the sake of brevity. A full description of the computation is shown in figure 2

The Buyer HTTP agent is described in table 6.The definition of the process is parametric on the URI of the Product resource to buy (μ_p). The Buyer agent creates a new PurchaseOrder resource through the URI μ_{pos} in the Seller application using a POST request. The pattern sent in this request (p_{no}) includes the URI of the product to be purchased and a literal for the state of the order with value "created".

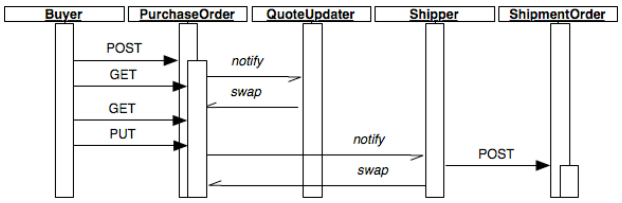

Figure 2: Sequence diagram for the modeled transaction

$$
\begin{array}{rcl}
Buyer(\mu_p) & ::= & CreateOrder.PollQuote \\
CreateOrder & ::= & \overline{req(\mu_{pos})}[post, p_{no}, 0]. \\
 & & [201, \mu_{po}]resp(\mu_{pos}) \\
PollQuote & ::= & \overline{req(\mu_{po})}[post, p_q, 0]. \\
 & & [200, q]resp(\mu_{pos}). \\
 & & if\ q\ ?\ ProcessOrder.PollQuote \\
ProcessOrder & ::= & \overline{req(\mu_{po})}[get, p_{mp}, 0]. \\
 & & [200, vq].resp(\mu po). \\
 & & if\ vq\ ?\ AcceptOrder.RejectOrder \\
AcceptOrder & ::= & \overline{req(\mu_{po})}[put, p_{mp}, accept]. \\
 & & [200, x]resp(\mu_{po}).0 \\
RejectOrder & ::= & \overline{req(\mu_{po})}[delete, 0, 0]. \\
 & & [200, x]resp(\mu_{po}).0
\end{array}
$$

Table 6: Buyer application processes

After the order is created and the URI for the new PurchaseOrder resource is returned μ_{po}, the Buyer process starts polling the newly created resource using GET requests containing a triple pattern that tries to retrieve the triples for the resource μ_{po} with an state of "quoted" (p_q). When one of these GET requests returns successfully with some triples, the Buyer agent issues a new GET request trying to retrieve the triples for the PurchaseOrder resource identified by μ_{po}, with price minor or equal to literal value min_price, using the pattern p_{mp}.

If this last GET request fails, the Buyer agent rejects the quote deleting the resource with a *delete* request. On the other hand, if some triples are returned, the Buyer process accepts the quote issuing a PUT request to the PurchaseOrder process, updating in this way the state of the resource to the value "accepted".

The Seller web application, described in table 7, contains two RESTful semantic resources Products ($Prods_{rest}$) and PurchaseOrders ($PurchOrds_{rest}$). Both of them have their associated triple spaces (θ_p, θ_{po}). These triple spaces are also manipulated by two agent processes QuoteUpdater ($QUpd$) and Shipper ($Ship$). QuoteUpdater gets a notification after each write operation of triples matching state "created" (p_{no}). Then, it goes through an internal transition τ that generates a price for the product in the PurchaseOrder and,

$$
\begin{aligned}
Prods &::= R_{rest}(\mu_{ps}, \theta_p) \\
PurchOrds &::= R_{rest}(\mu_{pos}, \theta_{po}) \\
QUpd &::= notify(\theta_{po}, write, p_{no}).pr ::= \tau. \\
&\quad swap(\theta_{po}, p_{no}, << p_{no}, quoted, > pr >) \\
&\quad .QUpd \\
Ship &::= \overline{notify}(\theta_{po}, write, p_{ok}). \\
&\quad \overline{req}(\mu_s)[post, p_{pok}, 0]. \\
&\quad [201, so]resp(\mu_s) \\
&\quad swap(\theta_{po}, p_{pok}, so).Ship
\end{aligned}
$$

Table 7: Seller web application

$$
Shipms ::= R_{rest}(\mu_s, \theta_s)
$$

Table 8: Shipment web application

finally, updates its triples with the price and a state of "quoted".

The Shipper agent gets notified when a new set of triples for a resource with state "accepted" (p_{ok}) is written in the PurchaseOrders triple space. After receiving this notification, it issues a POST request to the ShipmentOrders resource and updates the state of the PurchaseOrders resource with the state of "shipped" and a reference to the newly created ShipmentOrders resource.

Finally, The Shipment web application just contains a REST resource for the ShipmentOrders resources ($Shipms$). The formalization of the Shipment web application is shown in table 8.

6. CONCLUSIONS AND FUTURE WORK

RESTful semantic web services have the potential to enable a new generation of distributed applications, retaining the scalable and successful architecture of today's web, and adding the powerful data description and interoperability capacities of semantic data.

With this paper we have tried to provide a precise definition of a certain theoretical model for this kind of computation, combining two well known formalisms: tuple space computing and process calculi.

We have found that both formalisms are specially well suited for describing different aspects of RESTful semantic computation: triple space computing as a way of describing computations taking place inside computational loci, like web applications, that will be exposed to external processes as RESTful resources, and process calculus for the description of the exchange of HTTP messages between HTTP agents across different computational loci.

In our conception, RESTful semantic services are just processes receiving HTTP messages through a well known URI and manipulating an associated triple space according to the messages received and REST semantics.

As an example of how this calculus can be used for describing actual computations, a basic example from the literature on web services orchestration has been formalized in terms of a set of RESTful semantic web services and agents.

Further work on the calculus should deal with different aspects not presented in this article.

A type theory for the calculus must be developed. Ontology languages like OWL, rooted in the theoretical background of description logics, make possible the introduction of type systems dealing with well typed patterns and values associated to URIs and RESTful resources.

In its current form, the calculus describes computations as different sets of equations for each party taking part in the interaction. A global calculus for the description of the computation [19] must be defined, taking into account the inherent duality of the message passing operations introduced in the present formalism.

Furthermore, the message passing portion of the calculus lacks some features present in the triple space portion, like the presence of blocking operations, or a notification mechanism. Research must be taken in the different ways of extending REST semantics to allow this kind of coordination primitives in HTTP agents, communicating with each other through a shared RESTful resource.

A current implementation of the introduced calculus, built on top of technologies like Erlang OTP, RabbitMQ AMQP queue broker and the Open Sesame triple repository is currently being developed. In this implementation extensions of HTTP operations with blocking semantics, notifications and a subscription mechanism based in the websockets [20] standard have also being tested as part of our ongoing research on RESTful semantic web services.

7. REFERENCES

[1] Fensel D. et alt. (2006) Enabling Semantic Web Services: The Web Service Modeling Ontology Springer-Verlang.

[2] Fielding R. (2000) Architectural Styles and the Design of Network-based Software Architectures University of California, Irvine

[3] Kopecky, Vitvar & Fensel. (2009) hRESTS and MicroWSMO CMS WG Working Draft

[4] A. P. Sheth & K. Gomadam & J. Lathem. (2007) SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups IEEE Internet Computing, pages 91-94

[5] Fensel. (2004) D. Triple-space computing: Semantic Web Services based on persistent publication of information IFIP Internation Conf. on Intelligence in Communication Systems, pages 43-53

[6] Adida & Birbeck. (2008) RDFa in XHTML: Syntax and Processing W3C Recommendation

[7] R. Milner (1999) Communicating and Mobile Systems: the Pi-Calculus Cambridge University Press

[8] R. Milner & J. Parrow & D. Walker (1989) A calculus for mobile processes

University of Edinburgh
[9] R. Milner (1991) The Polyadic Pi-Calculus: a Tutorial
Logic and Algebra of Specification
[10] P. Hayes. (2004) RDF Semantics
W3C Recommendation
[11] D. Gelernter. (1985) Generative communication in
Linda
ACM Transactions on Programming Languages and
Systems, vol 7, pages 80-12
[12] N. Busi & R. Gorrieri & G. Zavattaro. (2000) Process
Calculi for Coordination: from Linda to JavaSpaces
Proc. of AMAST, LNCS 1816, pages 198-212
[13] A. Neves & E. Pelison & M. Correia & J. Da Silva.
(2008) DepSpace: A Byzantine fault-tolerant
coordination service
Proceedings of the 3rd ACM SIGOPS/EuroSys
European Conference on Computer Systems -
EuroSys, pages163-176
[14] A. Seaborne & E. Prud'homeaux. (2008) SPARQL
Query Language for RDF
W3C Recommendation
[15] E. Simperl & R. Krummenacher R. & L. Nixon. (2007)
A Coordination Model for Triplespace Computing
9th International Conference on Coordination Models
and Languages, pages 1-8
[16] D. Gorla (2006) On the relative expressive power of
asynchronous communication primitives
Proceedings of 9th International Conference on
Foundations of Software Science and Computation
Structures, vol 3921, pages 47-62
[17] N. Busi & R. Gorrieri & G. Zavattar. (2000) On the
Expresiveness of Linda Coordination Primitives
Information and Computation
[18] Kavantzas & Burdett & Ritzinger & Fletcher & Lafon.
(2005) Web Services Choreography Description
Language Version 1.0
W3C Candidate Recommendation
[19] M. Carbone & K. Honda N. Yoshida & R. Milner &
G. Brown & S. Ross-talbot. (2006) A theoretical basis
of communication-centred concurrent programming
[20] I. Hickson. (2010) The Web Sockets API
W3C Editor's Draft