Modeling Mash-Up Resources

Iván Pérez[2], Ángel Herranz[1], Susana Muñoz[1], y Juan José Moreno-Navarro[1,2]

[1] Babel Research Group
Universidad Politécnica de Madrid
{aherranz,susana,jjmoreno}@fi.upm.es
[2] IMDEA Software
ivan.perez@imdea.org

Resumen. Web-based applications to assemble web resources, like Yahoo Pipes or Netvibes, are emerging as a basis for a new wave of Internet applications that bring semantic web to the general public. Nevertheless, technologies and conceptual tools that allow ordinary users to combine application functionality are immature. There are some formal models but most of them are ontologies that do not capture the complex semantics behind resource composition and resource functionality. In this work we present a formal semantics, in the form of an ontology, to represent resources, composition of resources and resources in execution. Our semantics offers a framework for building tools that assist users to discover resources and connecting them depending on their functionality and their data types they share.

Palabras clave: Formal Methods, Component-Based Software, Service-Oriented Architectures, Data Modelling, Semantic Web, Ontology, Mash-ups.

1 Introduction

While the arrival of Web 3.0 will allow spreading the use of the Internet on more and more areas, it is still an issue how non-expert ordinary users can actively interact with it, more than merely be an information sink [15]. Novel Web-based applications referred to as *mash-ups* are currently emerging. Mash-ups allow human users to assemble, use and share Web-based resources (*gadgets*). Well-known examples are Yahoo Pipes [27], Netvibes [22] and IBM's QEDWiki [25], although many more are under active development.

These platforms lack a formal background to describe resources as well as to establish assembly and discovery criteria. This results into no support for automatic reasoning about composition, sharing and discovery of resources. Technologies and conceptual tools that allow users to combine resource functionality in a seamless, intuitive and efficient way is an area of research.

Easy Web

Suppose a catalogue of available Web-based resources. In the catalogue there is a gadget (web-based application with graphical view) that allows the user to choose an element from a given list. Other gadget in the catalogue receives two addresses and a departure time, and shows a map with the route between both addresses and the expected arrival time. A third Web-based application in the catalogue provides an address given the name of a school of the Technical University of Madrid (UPM).[3]

[3] UPM schools are spread all over Madrid.

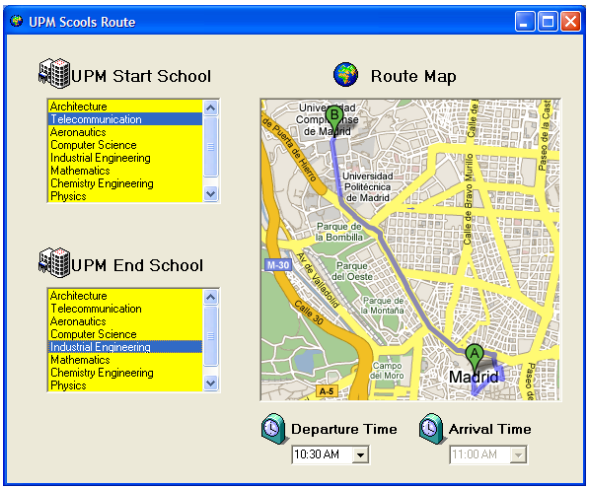

Fig. 1. Gadget with routes between UPM schools

Suppose a UPM officer is interested in a software application that offers routes between two UPM schools and the arrival time given a departure time. The officer will need to discover resources in the catalogue and compose his own gadget: Figure 1.

Parts of this guiding example will be formalised along the paper. The formal model we propose is an essential element of the EzWeb platform. The EzWeb project[4] aims at providing the architecture and implementation of an open standard web collaboration platform for building, composing and sharing web applications. EzWeb is, according to [15], a *mash-up* platform, although it goes significantly beyond the current state of the art, bringing together the concept of gadget and web-service-based SOAs.

This project should bridge the gap between technical representations of Web-based resources and the ordinary users and their informal, flexible way of creating and using these resources. High level requirements of the platform are:

- Users can get resources that provide access to contents, services and graphical interfaces.
- Users can customise their operating environment in a flexible and dynamic way, introducing and removing available resources and describing interactions among them.
- Users can program their own resources.
- Users can define new resources by composition of resources.
- Users can share resources.
- Users can share knowledge about the use of resources and their relationships.
- Users can discover relevant resources in the community.

Resource discovery based on textual description does not fulfil even the lowest requirements. Resource composition without any kind of type information does not allow reasoning about the compatibility of composition. A formal description of resource functionality is essential to provide a useful and reliable discovery and composition framework as well.

Our main contribution is a set of ontologies [16] as a formal model for defining, discovering, composing and exposing components of mash-up architectures. Each ontology has been written in OWL [14] because it is considered as a popular notation to represent concepts and information ordinary users are interested in. The formal model is presented in Section 3 as an iterative process: basic resource model (Section 3.1), resource composition model (Section 3.2) and resource execution model (Section 3.3).

[4] EzWeb is an industrial project led by Telefónica I+D with the participation of additional partners (http://forge.morfeo-project.org/wiki/index.php/Plataforma_EzWeb).

2 EzWeb Platform

The EzWeb [24] platform has revisited SOA (Service Oriented Architectures [3, 5]) to focus on end-users as service clients by providing a RESTful [7] data front-end, transforming the enterprise legacy information technology into a uniform layer of human-accessible resources.

Resources. The main concept in the EzWeb platform is *resource*. Resources are introduced as an abstraction layer between the users and the internal representation of data and programs. Features of resources are:

- Resources are simple software components that follow the *REST* paradigm [7]: they have a *URI* and respond to basic HTTP GET/POST/PUT and DELETE requests.
- When a resource encapsulates access to consult and modify external data sources we say it is a *content*.
- When a resource processes information from other resources we say that it is a *service*.
- Services are classified as *back-end services* and *operators*. Back-end services abstract the access to external services (probably legacy services in the form of web services). Operators merely transform data from other resources.
- When a resource produces information that can be integrated in the user interface through a graphical representation we say it is a *gadget*.
- Resources can be combined by composition of other resources.
- Composition and communication models must be simple enough to be used by end-users.

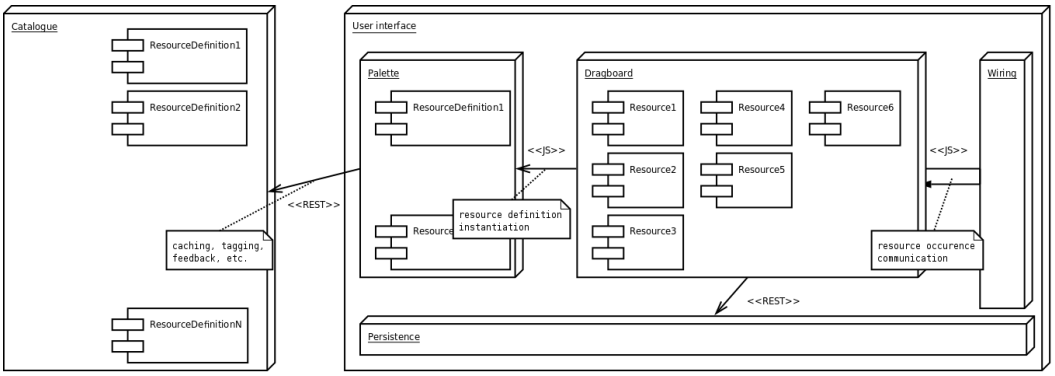

Fig. 2. EzWeb Reference Architecture

Reference Architecture. The EzWeb platform architecture is shown in Figure 2. The architecture allows to create and to compose resources through a friendly user interface. Resource definitions are stored in collaborative *Catalogue*s. Users can search for resource definitions in the catalogue and cache them in their *Palette*s.

Communication between Palette and Catalogue is *RESTful*. Users can *semantically* enrich the catalogue with usual Semantic Web techniques like assigning tags, values and recommendations that are shared with other users.

Once a resource definition is in the user's palette it can be instantiated and run in the *Dragboard*. Dragboards are configurable grids where resources run. Resource layout and preferences in dragboard can be modified by the user. Different instantiation of the same resource definition is allowed. Users can establish connections between internal *variables* of resources and the *Wiring* component allowing resources to communicate with each other. Resources and the Wiring component communicate by using a JavaScript API. The *Persistence* component allows resources in the Dragboard to be stored until the next session. Communication between Dragboard and the Persistence component is RESTful.

Although not shown in Figure 2, development tools in the platform allow programmers and users to create new resource definitions by directly programming them or by composing already defined resources in the catalogues. Once a new resource has been defined this can be added to the catalogue through its RESTful interface.

Informal Semantics. Before diving in the formal semantics we summarise the most relevant pieces of knowledge presented so far:

- The term *resource* sometimes refers to a definition of a kind of resource (*resource definition*), sometimes refers to an instance in execution of a resource (*resource occurrence*).
- Each resource definition has context information like author, vendor, recommendations, persistent state, rendering, context data version, creation date, preferences, description, image, author's email, data wiring, author's name, wiki. This kind of information is perfectly modelled by well known vocabularies and ontologies in the Semantic Web like Dublin Core [19], SIOC [26], FOAF [20] or DOAP [18] and will not be done explicit in our semantics.
- Data interchanged between resource occurrences is typed. This avoids runtime errors, helps in the discovering of resources and coherent communication points between resources.
- Resources can be created by just programming (simple resources) them or by composition of already defined resources (compound resources).
- Resource taxonomy must include contents, services and gadgets and must distinguish composed and simple resources.
- Two different communication models exist in the platform, one to communicate resources that are components of a resource defined by composition. The other to communicate resources running in the dragboard, i.e. resources in execution.

3 Semantics

In this section we present a formalisation of resources in the EzWeb platform as a set of ontologies [16] encoded in OWL [14]. OWL is based on Description Logics [2]. We assume the reader is acquainted with the syntax and the meaning of formulae of Description Logics.

We will present the semantics in an incremental way. In the first iteration (Section 3.1) resource definitions are formalised. More complex elements of the model and architecture presented in Section 2 like resource occurrences (resources in execution) and resource composition are introduced in subsequent iterations, Section 3.3 and Section 3.2, respectively. In each iteration, a new ontology is introduced.

Different terminologies are used to refer to ontology constructs. In this paper we follow Protégé [23] terminology (classes, properties, individuals, etc.) since the whole formalisation process has been done using release 3.3.1 of this ontology editor.

To avoid confusion due to name clashing, namespace prefixes are associated to each ontology and used whenever necessary. Those namespace prefixes are explicit in the title of each iteration (`def`, `rt` and `comp`).

We invite the reader to explore our formalisation with Protégé, `.owl` files with the ontologies can be downloaded from: http://babel.ls.fi.upm.es/formalspecs/owl/ezweb/1.0.

3.1 Iteration: Resource Definitions (`def`)

Three kinds of resources were mentioned in Section 2: contents, services and gadgets. Resource definitions will be individuals of class Resource. Each kind of resource is reflected as a different class in the ontology: Content, Service and Gadget.

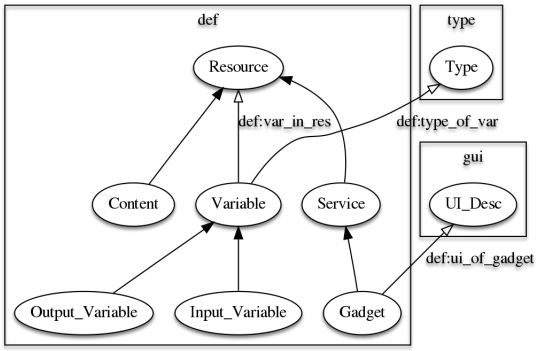

Fig. 3. Semantics of resource definitions

The following axioms capture disjointness of kinds of resource definitions and that no other resources definitions exist, why GADGET is a subclass of SERVICE will be apparent soon:

$$\begin{aligned} \text{RESOURCE} &\equiv \text{CONTENT} \sqcup \text{SERVICE} \sqcup \text{GADGET} \\ \bot &\sqsubseteq \text{CONTENT} \sqcap \text{SERVICE} \\ \text{GADGET} &\sqsubseteq \text{SERVICE} \end{aligned}$$

Resource definitions are templates for creating resource occurrences (resources in execution). Resource occurrences communicate with other resource occurrences by connecting internal variables. At this step we will capture that resources have got variables. Variable definitions play a similar role to formal parameters and are individuals of the class VARIABLE. VARIABLE is classified in the disjoint classes INPUT_VARIABLE and OUTPUT_VARIABLE:

$$\begin{aligned} \text{VARIABLE} &\equiv \text{INPUT\_VARIABLE} \sqcup \text{OUTPUT\_VARIABLE} \\ \bot &\sqsubseteq \text{INPUT\_VARIABLE} \sqcap \text{OUTPUT\_VARIABLE} \end{aligned}$$

Variables must be typed in order to avoid errors and to detect compatibility of variables when connected. Class TYPE is introduced. The semantics of this class has to be studied in depth so a new ontology is created: `type`. Types are defined, so far, as URIs that identify DTDs and XML Schemas that describe the format of data in resources. This decision is very easy to integrate in our ontology but we plan to design a more powerful type system in order to improve reasoning about data compatibility.

Every individual of VARIABLE has got an associated type. The functional total property type_of_var captures the association. The following axioms formalise, respectively, domain, range, being a total function (every variable has a unique type and at least one)[5]:

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{type\_of\_var}^{-1}.\text{VARIABLE} \\ \top &\sqsubseteq \forall \mathsf{type\_of\_var}.\text{TYPE} \\ \text{VARIABLE} &\sqsubseteq = 1\ \mathsf{type\_of\_var}.\top \end{aligned}$$

All variable definitions belong to one and only one resource definition. This relation is represented with the functional total property var_in_res.

[5] We will not repeat this common axiomatization scheme for other functional total properties.

Contents follow the REST paradigm and thus they have four possible input variables: GET, PUT, POST and DELETE and one output variable (the result of executing any method). Service definitions have any number of input and output variables. The following formula restricts the number of variables of a content to five:

$$\textsc{Content} \sqsubseteq = 5\ \mathsf{var\_in\_res}^{-1}.\top$$

To be precise, five of the variables must be individuals of Input_Variables. To capture this assertion we have had to introduce subproperties of var_in_res. An interested reader can check it at http://babel.ls.fi.upm.es/formalspecs/owl/ezweb/1.0/def.owl.

Gadgets can be regarded as services with a user interface. There is no task that a service can perform and a gadget cannot. This explains why Gadget is a subclass of Service. Gadgets have got a property that plain services do not have: the graphical interface. Therefore, we will also add an UI_Desc class to represent data related to the user interface. We put this class in a different ontology called `gui` because we expect it to become bigger in the future. So far this class is only related to gadgets (the only type of resources with user interface) through the functional total property ui_of_gadget.

A graphical representation of the ontologies seen so far is shown in Figure 3. Note that the black (non-empty) arrows indicate subclassing and therefore have no label. Empty arrows represent functional relations.

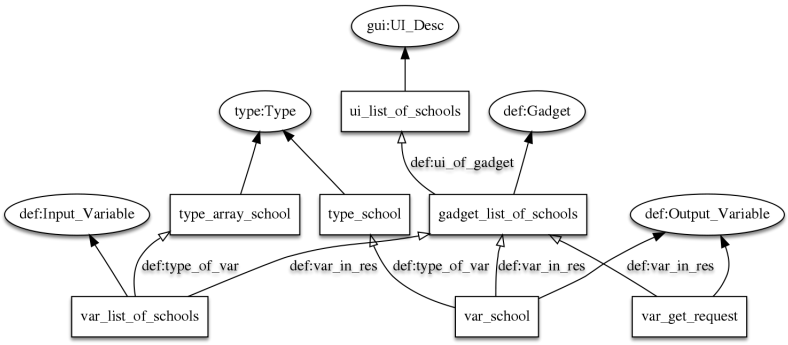

Fig. 4. A gadget definition

Example: Simple resources. With this ontology we can define EzWeb resources (contents, services and gadgets) and give some information about their interface: what variables they have and their type. We show a few examples of resource definitions that can be specified with semantics presented in this section:[6]

- Resource definition *content_list_of_schools* is an individual of class Content that defines contents that obtain a list of schools of a specific university from an external source (a figure with this resource not included due to lack of space).
- Resource definition *gadget_list_of_schools* in Figure 4 is an individual of Gadget that defines gadgets that present a list of schools to the user and, when one element of that list is clicked on, put that school's corresponding data to an output variable.
- A Service, *service_school_address_extractor*, defines services that extract the address from the data of a school (a figure with this resource not included due to lack of space).

[6] Individuals in figures are represented with boxes and white arrows represent instances of relations (we have abused the notation by using the same names for the individuals of properties and properties).

3.2 Iteration: Composing Resources (comp)

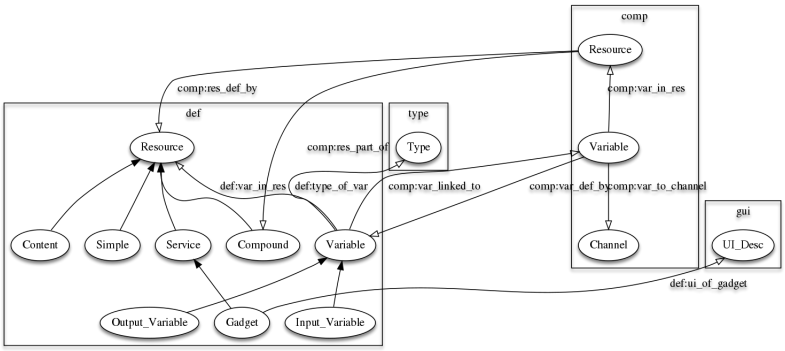

Fig. 5. Semantics of resource composition

In order to support resource composition, we classify resource definitions in simple and compound definitions. Classes SIMPLE and COMPOUND are disjoint subclasses of `def:`RESOURCE but every resource definition is in one of those classes:

$$\begin{aligned} \texttt{def:}\textsc{Resource} &\equiv \textsc{Simple} \sqcup \textsc{Compound} \\ \bot &\sqsubseteq \textsc{Simple} \sqcap \textsc{Compound} \end{aligned}$$

Composed resource definitions are the result of the composition of other resource definitions. We may be tempted to create a property with domain and range in `def:`RESOURCE. A richer model is needed, since we might want to include the same resource definition twice in a composition and distinguishing them is crucial. In order to make this possible, we introduce the concept of *component*.

Every component follows a resource definition. The new class `comp:`RESOURCE represents this concept in the ontology. The relation between a component and its resource definition is denoted by the functional and total property res_def_by.

Components of a resource definition will be captured by the property composed_by with domain COMPOUND and range COMPONENT. The inverse of this property is a functional total property named res_part_of:

$$\mathsf{composed\_by} \equiv \mathsf{res\_part\_of}^{-1}$$

To be composed by components is the expression that defines the class COMPOUND as reflected by this axiom:

$$\textsc{Compound} \equiv \geq 1\ \mathsf{composed\_by}.\top$$

Although Figure 5 is a graphical representation of the complete semantics in this iteration, the reader can check semantic elements presented until now.

We capture now how variables of components are connected in a compound definition. Individuals representing variables in components are needed and a new class `comp:`VARIABLE is created. The functional total properties `comp:`var_def_by and `comp:`var_in_res relate individuals of `comp:`VARIABLE, respectively, to their definitions in `def:`VARIABLE and to the resource definition they belong to.

In our model, component variables are not directly linked to other variables. *Communication channels* are used to link component variables. Those channels are individuals of

class COMP:CHANNEL and functional total property var_to_channel denotes that a component variable is connected to a channel. An essential property of the model we are interested in is that component variables connected to a channel must have the same type. In this case, and many others, we need property composition as construct in the logic. Nevertheless, composition would lead to undecidable assertions.

A subtle problem is how variables in the definition of a compound resource are connected to component variables. There are several strategies to tackle this situation, and although some of them may seem appropriate at first, they end up causing some problems.

A solution we tried was to treat every component variable not attached to a channel (i.e. $\neg(\exists \text{var\_to\_channel}.\top)$ as a variable of the compound resource definition. This means that def:VARIABLE $\sqcap$ comp:COMPVAR is a consistent class. Elegant but problematic: variables of two components related to the same compound resource definition through property res_def_by cannot be distinguished.

Our solution is to explicitly introduce variable definitions for the compound definitions as done for any other resource definition and link them to component variables not attached to any channel. This decision gives us the flexibility to decide what variables are not necessary in a compound resource and can be hidden to avoid unexpected results.

According to this, we introduce a functional property comp:var_linked_to from individuals of def:VARIABLE of a compound resource with individuals of comp:VARIABLE of an internal component. Note that neither this nor the inverse relation (which relates a component variable with a variable definition of the resource it is part of) are total.

Example: Resource composition. To show in practice how components can be represented and create a compound resource, we will show two components for two definitions given in the previous section. The objective is to create a compound gadget that shows a list of schools to the user without having to provide it through an input variable.

Figure 6 shows one of the components to be used (a figure with the gadget's content is not included due to lack of space). The gadget component has a variable called *var_get_request*. This variable is used to connect the gadget with the *var_get* input variable of the content. When information is sent through that channel, the content reads that information, performs a request to the external source of data and puts the result in the output[7]. The composition of these two resources to create a new *list_of_schools* gadget is shown in Figure 7. Note that, due to lack of space, we only include in each picture the classes and individuals that we see essential to help understand resource composition.

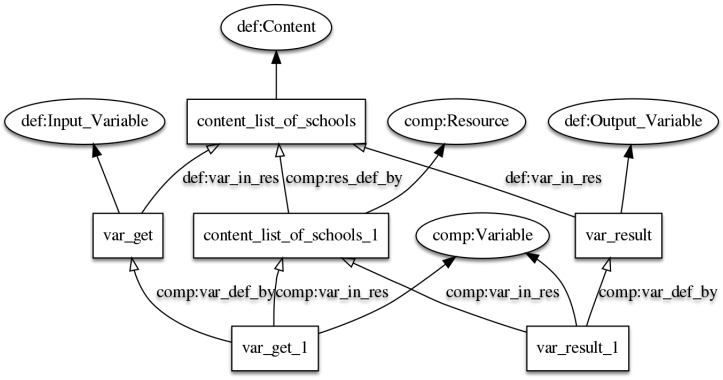

Fig. 6. A content component

[7] Contents do not output anything until a request to the external data source is submitted through one of its inputs. Content inputs may not carry significant data and just act as triggers that will command the content to submit some fixed message to the source.

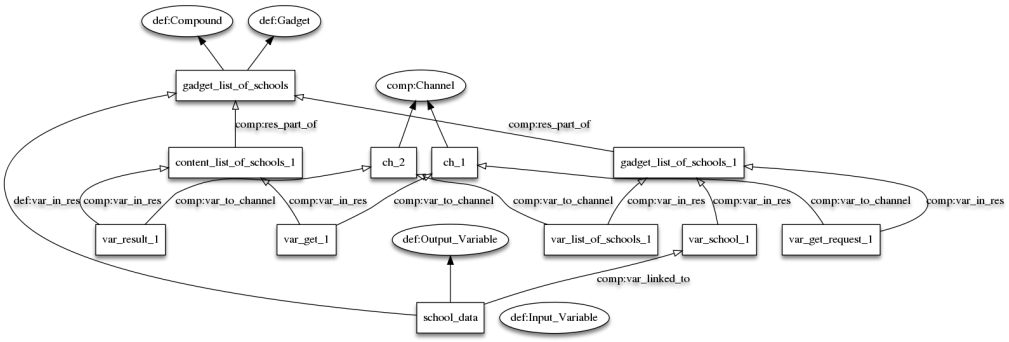

Fig. 7. A compound gadget

3.3 Iteration: Resource Execution (rt)

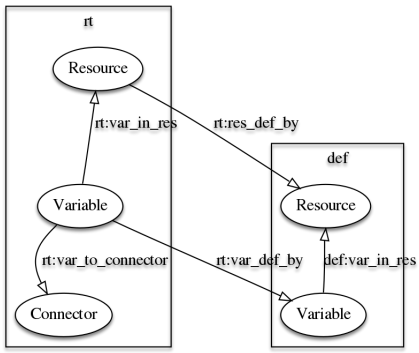

Fig. 8. Ontology representing resource occurrences in execution

This iteration is very similar to iteration of Section 3.2. When a resource definition is chosen by a user to run in the dragboard we have an individual of class rt:Resource. These individuals represent resources at runtime that we will call resource occurrences.

Obviously resource occurrences are defined by a resource definition and the functional total property rt:res_def_by reflects that part of the model. Let us show the axioms that formalise this information:

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{rt{:}res\_def\_by}^{-1}.\mathtt{rt{:}}\textsc{Resource} \\ \top &\sqsubseteq \forall \mathsf{rt{:}res\_def\_by}.\mathtt{def{:}}\textsc{Resource} \\ \mathtt{rt{:}}\textsc{Resource} &\sqsubseteq = 1\ \mathsf{rt{:}res\_def\_by}.\top \end{aligned}$$

Since resources can communicate in the dragboard, we need individuals to represent actual variables (rt:Variable) and individuals to connect those variables (rt:Connector). Connectors are similar to channels but introducing a new class allows us to specify a different communication model from the one used in composition of resources.

Properties relating new individuals follow patterns identical to those used in the iteration at Section 3.2:

- Functional total property rt:var_def_by relates individuals of rt:Variable in the dragboard with a individuals of def:Variable.
- Functional total property rt:var_in_res relates individuals of rt:Variable with the individuals of rt:Resource they belongs to.

– Functional property `rt:var_to_connector` relates individuals of `rt:`VARIABLE with individuals of RT:CONNECTOR. This means that the variable is attached to that connector[8]. This function is not total: a variable does not necessarily have to be connected to anything.

A graphical representation of the classes and properties defined in this section is presented in Figure 8.

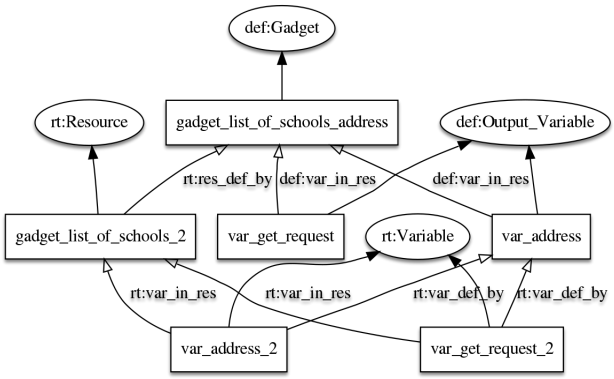

Fig. 9. Individual that represents a gadget at runtime

Example: Resource Execution. In Figure 9 we show the compound gadget created in Section 3.2. Representing connections of resources in the dragboard is simpler than in composition since connection does not yield compound resources. Figure 10 shows that only one new individual and two property instances are needed to represent this connection between runtime variables.

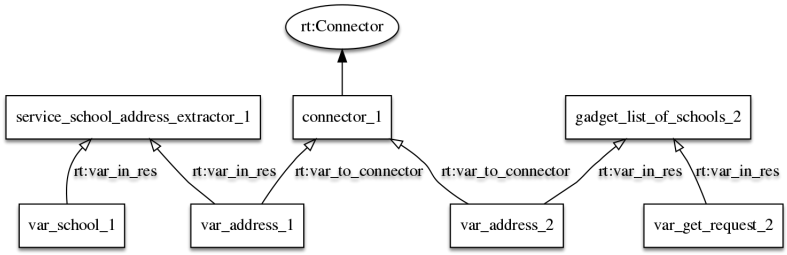

Fig. 10. Individuals that represents a gadget and a service connected at runtime

4 Related Work

To the best of our knowledge there is no other attempt to give a formalisation of mashup concepts, beyond those that can be captured by well established ontologies like Dublin Core [19], SIOC [26], FOAF [20] or DOAP [18].

Much more ambitious aims can be found in the area of formalisation of Web services (WSDL [4], OWL-S [13], WSMO [12], SAWSDL [6]), grid services [8], business processes (BPEL [1]) and service orchestration (WSCDL [10] and Linda [28]).

Lightweight variants of Web service formalisations like WSMO-Lite [17] and the work on SA-REST [11] for expressing semantics of RESTFul services are more similar to our formalisation aims. We find their abstraction level too close to the implementation. In particular, they manage concepts like data mediation and protocols for service invocation that we consider inappropriate for end-users.

[8] Whether it writes or reads from the connector depends on the kind (input or output) of variable it is.

5 Conclusions and Future Work

We have formalised the resources of a mash-up platform in OWL: resource definition (templates to instantiate resources in execution), resource composition and communication and resource runtime information. We have presented our formalisation methodologically, following an iteration process. Each iteration introduced new ontologies. Complex ontologies have been built on top of simpler ones. The structure of our formalisation lets us change semantics at different abstraction levels. The use of a method and the success of OWL as a popular formal notation turn our model into a formal model of mash-up platforms that is easy to understand.

In spite of this simplicity, the semantics allows reasoning about resource definition, discovery, composition and exposure. For the moment, we have used KAON2 [9, 21] to identify empty classes and inconsistencies in redundant definitions, and to deduce properties that go unnoticed.

From a more practical point of view, every individual of the ontology corresponds to an actual object in the platform. Individuals of `def:`Resource are objects in the catalogue. Individuals of `rt:`Resource are programs running in the user front-end. Individuals of `rt:`Connector are *pipes* in the wiring module, etc. We claim that OWL could be used as the concrete interchange format for most elements of the platform. Other agents in the Semantic Web could understand EzWeb resources and eventually use them. Furthermore, automatic and well-studied reasoners could be directly used in powerful discovery processes started by users.

OWL-Lite and OWL-DL are not expressive enough to capture some interesting properties of our model. For instance, constraints on property composition are needed to keep information consistent between definition and runtime levels. OWL-Full or a more expressive description logic can be used, taking into account that the decidability of the deduction process will be compromised.

Future work. The semantics of types and communication models are very loose. We plan to refine the following classes to improve the formalisation:

- Type. XML definition formats like DTDs or XML Schemas are not enough to capture polymorphic data types that make output data depend on input data. The Formalisation of type systems that support subtyping, parametric polymorphism, or even dependent types will be needed.
- Channel and Connector. A taxonomy of these classes is needed to express different communication models. This taxonomy will capture how channels are named and referred to, if they are synchronous or asynchronous, if they are bounded or not, if they can be accessed by multiple resources, etc.

We think that the resource implementation (programs) can be automatically synthesised from resource definitions. Code could be synthesised from resource definitions if a notation for behavioural specifications is included in the semantics. Another, almost immediate alternative is to generate code for compound resources once formal communication models are established.

Acknowledgements

The work reported on the paper has been developed in the framework of the Morfeo EzWeb strategic singular project, funded by Ministry of Industry, Tourism and Trade under grant FIT-340503-2007-2 and an agreement between Telefónica I+D and IMDEA Software. It is also partially supported by research projects DESAFIOS (Ministry of Science and Innovation TIN2006-15660-C02-02) and PROMESAS (Comunidad de Madrid S-0505/TIC/0407)

Referencias

1. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services version 1.1. Technical report, IBM, May 2003.
2. Franz Baader, , Ian Horrocks, and Ulrike Sattler. *Mechanizing Mathematical Reasoning*, volume 2605/2005 of *Lecture Notes in Computer Science*, chapter Description Logics as Ontology Languages for the Semantic Web. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-25051-7, ISSN 0302-9743 (Print) 1611-3349 (Online).
3. Michael Bell. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, chapter Introduction to Service-Oriented Modeling. Wiley & Sons, 2008.
4. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. Note, W3C, March 2001.
5. Thomas Erl. *Service-oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River. Prentice Hall PTR, 2005.
6. Joel Farrell and Holger Lausen. Semantic annotations for wsdl and xml schema. Recommendation, W3C, August 2007.
7. Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
8. Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
9. U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics with a concrete domain in the framework of resolution. In *Proc. of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 353–357, Valencia, August 2004.
10. Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. Candidate recommendation, W3C, November 2005.
11. Jon Lathem, Karthik Gomadam, and Amit P. Sheth. SA-REST and (S)mashups : Adding semantics to RESTful services. In *Proc. of International Conference on Semantic Computing*, 2007.
12. Holger Lausen, Axel Polleres, and Dumitru Roman. Web service modeling ontology (WSMO). Member submission, W3C, June 2005.
13. David Martin. OWL-S: Semantic markup for web services. Member submission, W3C, November 2004.
14. Guus Schreiber and Mike Dean. OWL web ontology language reference. Recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.
15. Christoph Schroth and Oliver Christ. Brave new web: Emerging design principles and technologies as enablers of a global SOA. In *IEEE SCC*, pages 597–604, 2007.
16. John F. Sowa. Ontology, metadata, and semiotics. In *ICCS '00: Proceedings of the Linguistic on Conceptual Structures*, pages 55–81, London, UK, 2000. Springer-Verlag.
17. Tomas Vitvar, Jacek Kopecký, Maciej Zaremba, and Dieter Fensel. Wsmo-lite: Lightweight semantic descriptions for services on the web. In *ECOWS*, pages 77–86, 2007.
18. Description of a project (DOAP). http://trac.usefulinc.com/doap/.
19. Dublin core (DC) metadata initiative. http://dublincore.org/.
20. The friend of a friend (FOAF) project. http://www.foaf-project.org/.
21. KAON2 ontology management for the semantic web. http://kaon2.semanticweb.org/.
22. Netvibes. http://www.netvibes.com.
23. Protégé ontology editor. http://protege.stanford.edu/.
24. Proyecto EzWeb. http://ezweb.morfeo-project.org/.
25. Qedwiki. http://services.alphaworks.ibm.com/qedwiki/.
26. Semantically-interlinked online communities (SIOC). http://sioc-project.org/.
27. Yahoo! pipes - editing pipe. http://pipes.yahoo.com/pipes/pipe.edit.
28. George Wells. Coordination languages: Back to the future with linda. In *Proc. of 2nd. International Workshop on Coordination Languages and Adaptation Techniques for Software Entities, WCAT05*, 2005.