Rule-based Orchestration of Networked Components

Andreas Harth, Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany

Position paper for the Workshop on IoT Semantic/Hypermedia Interoperability, July 2017, Prague, Czech Republic

Abstract

The paper presents challenges concerning the integrated access to sensors and actuators in networked environments. We draw on experience gained through data integration and system interoperation projects with both academic and industrial partners, and from lessons learned during the development of several prototoypes. We have identified architectural mismatches that require mapping and integration before applications can access and manipulate the state of components in a uniform manner. We tackle mismatches along the following dimensions: network protocol, data format and data semantics. Our system architecture builds on ideas from Representational State Transfer and uses standards around Linked Data. For discovery we assume that network-accessible components provide hyperlinks to other components. On top of the standardised interfaces we use a rule-based language for integrating state, following links and specifying application behaviour.

Introduction

Modern software systems have to incorporate an increasingly diverse set of components, both in terms of hardware (sensors, actuators) and software (APIs, services). Many of these components are accessible via a network interface and run in a distributed setting. The heterogeneity of the components leads to a high cost for building applications. Performing the integration in a monolithic application requires glue code to access and manipulate the state of each component. Similarly, performing the integration in a distributed application also requires glue code in the form of wrappers to access and manipulate the state of each component.

A way to reduce the overall amount of manual integration effort is to provide standardised interfaces to components, and to reuse the standardised interfaces in multiple applications. When integrating different interfaces, one can employ different strategies for deciding on the features of the common interface. One strategy is to use the union of the feature sets of the source interfaces; another strategy is to use the intersection of the feature sets of the source interfaces; yet another strategy is to pick and choose among the feature sets of the source interfaces. However, as the components use different, sometimes inherently incompatible, paradigms for accessing and manipulating state, the specification of uniform interfaces remains a challenge. In addition, the requirements for interfaces are very broad: interfaces should be simple and easy to implement and use, yet at the same time they should satisfy the requirements of very different scenarios. Often, the requirements of the various scenarios are not made explicit, as they are completely clear to anybody who is part of a particular community. However, once one tries to access interfaces of devices and components from different communities, the various unspoken assumptions cause problems.

We use a core idea from Web Architecture and Representational State Transfer to keep the uniform interface specification managable. We dictate constraints to limit the degree of freedom in the way that interfaces can be implemented. Minimal interfaces make cost-effective interoperation possible. At the same time, the interfaces should be sufficiently high-level and be building on a universal model for computation to put as little restrictions as possible on what can be theoretically achieved. There is always a trade-off, and different people have different tastes and styles. Our proposal is to build on standards that have proven to work in the past: internet technologies, as those have been successfully deployed on a global scale. However, we think that the abstraction the core internet technologies provide (basically a channel where packets can be send and received) is too low-level. Web technologies and the resource abstraction with request/response communication between user agents and origin servers provide a higher level abstraction, and that abstraction has been successfully deployed on the web.

Another useful feature of the web are hyperlinks to allow for decentralised publication and discovery. To leverage the full power of hyperlinks, which allow for applications to discover new things at runtime, the newly discovered things have to be self-described. In such an environment, applications could then follow hyperlinks and use arbitrary data and functionality from hitherto unknown components. At least that is the vision; the realisation of that vision has been proven to be challenging, not the least because developers have to create applications which operate on data with arbitrary schema unknown at design-time of the application. Semantics provides the means for at least some freedom for mapping and integrating data with different schema.

In the following we present the architectural mismatches we have identified in our work on academic and industrial data integration and system interoperation applications. We describe mismatches concerning network protocol, data format and data semantics that preclude applications from accessing data and functionality of components in a uniform manner. To overcome the mismatches, we assume a uniform addressing scheme (URIs), uniform network protocol (HTTP) and uniform data format (RDF), on top of which applications operate. The behaviour of applications is specified using rules. These rule-based applications can access and integrate resource state, follow links to discover new resources, and change resource state to implement application behaviour. While we have an eye on elaborate functionality such as artificial intelligence planning and model checking, the focus on our work so far has been on optimised execution of application behaviour specifications. Due to space constraints, we can only briefly introduce the rule-based language, but we provide pointers to further material.

Constraints

To reduce the effort for integrated access to component state, we assume the following constraints for the uniform interfaces to components:

- Each component provides resources, which are identified via a URI.
- Components provide a HTTP server interface and allow for CRUD operations on the provided resources using HTTP.
- The resource state is represented in RDF.
- Hyperlinks from a resource to another resources; at least links from an index resource to other resources on the same component.

The first two constraints relate to RMM level 2. RMM level 2 means that we view sensors and actuators as resources and identify the resources with URIs. Reading out sensor state is done via GET, and changing actuator state is done via PUT. The four constraints (minus the CRUD operations using HTTP PUT/POST, GET, PUT and DELETE on the second constraint) map to the Linked Data principles.

There is one active component per application; that active component is a HTTP user agents that poll resource state. The requirement for one user agent per application could be relaxed, but is useful in the beginning to reduce complexity. Also, one could add a server interface to the controlling component in an application. However, the same argument regarding reduced complexity applies for our decision to start with one active component per application.

Mapping Different Protocols

Because not all components implement the constrained interface, we require wrappers to bring the components to the same level. Next to syntax and semantics of resource representations (covered in the following sections), we might need to map fundamentally different networking protocol paradigms.

The constraint interface assumes a HTTP server interface access, with the components being passive and waiting for incoming requests.

Some networking environments assume active components, i.e., components that emit data at intervals. To be able to include those components in our system architecture, we require a wrapper that provides the current resource state via GET on HTTP URIs. That is, the wrapper has to receive the messages from an active component and store the resource state internally, and make the resource state (passively) accessible via GET on URIs. Analogously, changing resource state (via PUT, POST or DELETE) has to be converted to the appropriate messaging format and then sent on to the final destination.

We have implemented such an protocol mapping interface between the Robot Operating System (ROS) message bus and our interface abstraction based on Read-Write Linked Data. The potential drawback is that polling is out of sync with the arrival of events, and that the application might miss state changes due to a polling frequency that is out of sync with the update frequency.

Mapping Different Representation Syntax

Even if all components provide HTTP server interface, the representation of the data might still differ (for example, binary formats, CSV, TSV, JSON or XML). Hence, the wrapper (sometimes also called "administration shell" or "adaptor") needs to lift the data model of the different components to a common data model. We assume the RDF data model. Given that RDF can be serialised into different surface syntaxes, the wrappers can chose to support different serialisations, for example RDF/XML, JSON-LD or Turtle.

Mapping Different Representation Semantics

Semantics can be mapped most of the time (SSN to TD, for example). Please note that both SSN/SOSA and TD are still being specified, so any discussion related to these vocabularies could be outdated.

Time

Granularity

However, some ontologies have different assumptions. SOSA, for example, assumes that we want to model a way to represent a journal of sensing and actuating activities. So there is an sosa:Observation class, and a sosa:Actuation class. Instances of these classes represent results from observation and actuation events. The communication protocol is separate; the modellers assume implicitely some service-oriented architecture. Performing a GET on a resource that returns an instance of (the latest) sosa:Observation is fine. The result of a PUT (in SOA terminology the postcondition or effect) is an instance of sosa:Actuation, which could be sent in a response message to a PUT. However, it is unclear what the message body should be for a PUT request.

TD, on the other hand, has a more immediate state-based representation. State representations include the "writable" flag, which indicates whether a representation (such the current temperature reading or the state of a switch) can be written.

TD could be mapped, for example:

```
td:Thing rdfs:subClassOf ssn:Sensor .
```

Another mismatch on the level of semantics is that of aspect (related to linguistic aspect). Messages could be represented as current state (for example, lat/long of a person), or using a higher-level description (for example, stating that the person is walking from A to B). Integration of the different aspects (resource-based vs. event-based) is an open challenge.

Link Following with Safe Requests

Assuming the constained interface, we can write rules that follow links. We use condition-action rules encoded in Notation3 syntax. Notation3 is a superset of Turtle, and extends the RDF data model with variables and graph quoting, so that subject and object of triples can be entire graphs.

The following rule specifies that "next" links should be followed:

```
{ ?x :next ?next . } => { [] http:mthd httpm:GET;
                              http:requestURI ?next . } .
```

Such rules allow for specifying that certain links to URIs should be followed.

URI templates are another way to specify links:

```
{ ?x :lat ?lat ; :long ?long . } => { [] http:mthd httpm:GET;
                                         http:requestURI "http://example.or
```

In a sense, the description is in the rule heads.

The rule interpreter applies these rules to an initial dataset to exhaustion, that is, until now new data (or requests) can be generated any more and a fixpoint is reached.

Input/Output Descriptions

In earlier version of our prototypes we have included LIDS descriptions of the input and output of services. For example, the required parameter for GET requests (encoded in the URI), and the output of the response message. However, we wrote rules directly encoding the parameters in URI templates and did not make use of the descriptions. The descriptions, because they were manually constructed and did not serve a purpose (not even for generating documentation), soon became outdated, as developers changed the API but did not change the descriptions.

Similarly, we allow for unsafe requests in the head of rules:

```
{ ?x :next ?next . } => { [] http:mthd httpm:PUT;
                              http:requestURI "http://example.org/r/foo" ;
                              http:body { [] a :State ; :state :Off . } } .
```

Because our focus is on execution.

The people who write rules know the interface and the required payload, hence we do not need descriptions.

Another scenario emerged in one of our projects where partners wanted to interface with components in code. They needed to generate messages from programming language objects. To that end, the open world assumption in RDFS and OWL turned out to be problematic, as validation of incoming and outgoing messages was not possible (for serialisation and deserialisation). Hence, the vocabulary descriptions in RDFS and OWL were augmented with descriptions in SHACL.

Given that we specify rules directly and focus on efficient execution. The rules could also be generated using an AI planning approach, based on IOPE descriptions and a goal specification. However, due to the high investment in IOPE descriptions, we have reserved such approaches for future work. Immediate goal is to provide high-level executable specifications based on rules, which could also serve as basis for formal verification methods.

Conclusion

We have presented a system architecture that provides integrated access to networked components in conjunction with a rule-based language to access resource state, integrate data and specify application behaviour. We believe that the standardisation of interfaces can benefit from the knowledge of which applications are supposed to make use of the interfaces, and how these applications are going to be developed.

We have implemented the described interfaces and applications in several prototypes in the areas of geospatial data integration as well as industrial applications around product design and validation. With appropriate care in implementation, our prototypes achieve update rates sufficient for industrial requirements. For example, we have achieved update rates of around 30 Hertz in virtual reality/augmented reality applications. We believe that the encountered problems and the proposed solutions generalise to other application areas in the area of cyber-physical systems.

We have begun working on a formalism based on state transistion systems and mathematical logic to provide a rigourous foundation for accessing components and specifying application behaviour. The ultimate goal of the formalism is to provide the ability to formally check properties of application behaviour specifications. On top of the foundational formalism we can layer additional functionality, for example to model systems that detect conditions on integrated resource state described in RDF or to simplify application development for casual users based on a workflow language.

Acknowledgements

The paper is based on joint work with Tobias Kaefer, Felix Keppmann, Sebastian Speiser, Steffen Stadtmueller and Rudi Studer. We have to mention Ruben Verborgh who suggested the use of URI templates in rules. We benefitted from discussions with members of the W3C Spatial Data on the Web working group, especially from discussions with Maxime Lefrancios.

References

[Harth and Kaefer 2016] Andreas Harth, Tobias Käfer. "Towards Specification and Execution of Linked Systems". 28. GI-Workshop Grundlagen von Datenbanken, May 24 - 27, 2016, Nörten-Hardenberg, Germany.

[Harth et al. 2013] Andreas Harth, Craig Knoblock, Steffen Stadtmüller, Rudi Studer and Pedro Szekely. "On-the-fly Integration of Static and Dynamic Linked Data". Fourth International Workshop on Consuming Linked Data (COLD 2013). Co-located with ISWC 2013, Sydney, Australia.

[Harth et al. 2016] Andreas Harth, Tobias Käfer, Felix Leif Keppmann, Dimitri Rubinstein, René Schubotz, Christian Vogelgesang. "Flexible industrielle VT-Anwendungen auf Basis von Webtechnologien". VDE Kongress 2016, Internet der Dinge, Nov 7-8, 2016, Mannheim, Germany.

[Kaefer et al. 2016] Tobias Käfer, Sebastian Bader, Lars Heling, Raphael

Manke and Andreas Harth. "Exposing Internet of Things Devices on REST and Linked Data Interfaces". 2nd International Workshop on Interoperability & Open Source Solutions for the Internet of Things. Co-located with 6th International Conference on the Internet of Things (IoT 2016). Nov 7, 2016, Stuttgart, Germany.

[Keppmann et al. 2014] Felix Leif Keppmann, Tobias Käfer, Steffen Stadtmüller, René Schubotz and Andreas Harth. "High Performance Linked Data Processing for Virtual Reality Environments". International Semantic Web Conference (Posters & Demos). ISWC 2014, Riva del Garda, Italy.

[Keppmann2 et al. 2014] Felix Leif Keppmann, Tobias Käfer, Steffen Stadtmüller, René Schubotz and Andreas Harth. "Integrating Highly Dynamic RESTful Linked Data APIs in a Virtual Reality Environment". International Symposium on Mixed and Augmented Reality (Posters & Demos). ISMAR 2014, Munich, Germany.

[Stadtmueller et al. 2014] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, Rudi Studer. "Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data". WWW 2013, Rio de Janerio, Brasil. Please note that we have renamed the system to "Linked Data-Fu" to avoid name clashes with other projects.

Draft