Abstract—Large-scale Internet of Things (IoT) applications, like Smart Cities, are ever changing pieces of software. The paradigm of Entity-Component-Attribute (ECA) based software design is well-suited to keep these applications changeable and maintainable. At the same time, W3C working groups propose the Web as IoT convergence platform. Semantic annotations on the data in RDF help tackle cross-domain interoperability issues. While the generation of RDF data from data storage layers has undergone thorough research, the automatic exposure of application run-time data is to this day incomplete. We for this formalize the Entity-Component-Attribute pattern and present an automated mapping to a structure compliant with the Linked Data Platform W3C standard. This structural mapping is then augmented by application domain specific semantics. The result lifts a software design pattern highly suitable for large-scale IoT applications to Linked Data.

Keywords-Entity-Component-Attribute model; Linked Data; Resource Description Framework (RDF); Mappings;

I. INTRODUCTION

The continual change that software undergoes during its lifetime is generally called evolution, and the degree to which it is easy or hard to change existing software is often called changeability [1], [2]. Especially in perdurable and large scale Internet of Things (IoT) platforms, software design with the intention to optimize changeability is imperative [3]. Towards this end, the architectural pattern of Entity-Component-Attribute (ECA) based software design is particularly well-suited [4]. Focusing on the principle of “composition over inheritance”, the role of an entity is no longer determined by class inheritance or attribution hierarchy, but dynamically determined by the set of attached components. This significantly improves changeability of entities and reuse of components.

ECA patterns have been successfully applied in the design of changeable IoT platforms and applications [5], however, the enablement of seamless cross-domain interoperability between independently developed IoT applications and platforms, one of the central challenges facing IoT [6], is not directly addressed by this design paradigm.

In this respect, the W3C Web of Things Working Group¹ proposes to use the Web as an IoT convergence platform. The group develops initial standards for this Web of Things (WoT) [7] by defining a Web-based abstraction layer for IoT platforms, protocols, data models and communication patterns. To unleash its full potential, the emerging WoT is expected to evolve into a Semantic Web of Things (SWoT) [8].

The SWoT will heavily rely on Linked Data principles [9] to semantically describe IoT entities semantically in terms of their actions, properties, events and metadata [10] independent of the underlying IoT platform. For large-scale scenarios and environments, the development of IoT platforms and applications on the SWoT will require software tooling that enables

1) (semi-) automated mappings from application data layouts to Linked Data
2) declarative augmentation of Linked Data with application-specific semantics
3) exposition of dynamic IoT runtime data as Linked Data

While there are numerous works investigating (semi-) automated mappings from heterogeneous data structures and serializations to the RDF data model [11], [12], [13], [14], [15], little research has been conducted on the dynamic mapping of runtime environments [16], [17] to RDF. In particular, we are not aware of any works on how to leverage ECA-driven software applications on the Semantic WoT.

This paper makes the following contributions. We formalize the notation of an ECA system, and detail on the automated structural mapping between ECA runtimes and the W3C Linked Data Platform². Next, we explain how domain experts can declaratively augment these generated structural mappings with domain-specific semantics. Finally, Linked Data clients can retrieve data as provided by the ECA-based Web Application via a semantic Linked Data Web-interface.

II. ENTITY-ATTRIBUTE MODELS

The understanding of Entity-Attribute models varies in literature. Common for all variations is the notion of an entity as an empty data container which is closer specified by a set of typed attributes that carry the actual values. The
Let \( e \) denote an entity instance, and \( \mathcal{P}_C \) denote the set of all component prototypes. Then we define the following sets:

\[
\begin{align*}
\mathcal{E} &= \text{the set of all entity instances. An entity instance is defined as } e = (n_e, C_e), \text{ with } n_e \in \Sigma^+ \text{ being the unique identifier for } e \text{ over alphabet } \Sigma, \text{ and } C_e \text{ being the set of component instances attached to } e. \\
\mathcal{C}_e &= \text{the set of all component instances attached to an entity instance } e. \text{ A component instance is defined as } c = (n_c, p_c, \mathbf{A}_{c,e}), \text{ with } n_c \in \Sigma^+ \text{ being the unique identifier for } c, \text{ } p_c \in \mathcal{P}_C \text{ being the prototype that } c \text{ is an instance of, and } \mathbf{A}_{c,e} \text{ being the set of all attribute instances attached to } c. \\
\end{align*}
\]

III. LINKED DATA PLATFORM IN VERY BRIEF

The Linked Data Platform (LDP) specifies a simple architecture to expose Linked Data on the Web.

The basic element of LDP is a LDP Resource. Every LDP Resource must be an HTTP endpoint with at least HTTP/1.1 protocol compatibility, and accept at least HTTP GET requests, and others depending on the type of resource.

ldp:RDFSource exposes its data upon a GET request and MUST return a full RDF graph in text/turtle format (or application/ld+json, if requested).

ldp:Container is a ldp:RDFSource that manages a set of LDP Resources and provides information about access, modification, and filtering of the contained elements.

ldp:BasicContainer is a ldp:Container that specifies linked documents in the form of Containment Triples of the form \((\text{container-uri}, \text{ldp:contains}, \text{document-uri})\).

The LDP specification specifies more concepts. Our mapping, however, will only make use of above concepts.

IV. STRUCTURAL INTEROPERABILITY

Structural interoperability establishes a common format for data exchange between applications [21]. In the following, we detail on the automated structural mapping between ECA runtime environments and W3C LDP compliant Linked Data servers. By repeated application of a set of mapping rules (cf. \( \mathcal{R}_1 \) to \( \mathcal{R}_4 \)), structural interoperability between ECA runtimes and LDP servers is established.

We assume the existence of functions \( \nu : \Sigma^+ \to \text{IRI} \) and \( \rho : \mathcal{P}_C \to \text{IRI} \) for minting fresh IRIS from identifiers and component prototypes. Although several guidelines exist for minting IRIS\(^3\), we do not make assumptions on \( \nu \) or \( \rho \).

\[
\frac{(n_e, C_e) \in \mathcal{E}}{\nu(n_e) \text{ rdf:type } ldp:BasicContainer.} \\
\frac{\forall (n_e, p_c, \mathbf{A}_{c,e}) \in \mathcal{C}_e}{\nu(n_e) \text{ dct:identifier } "n_e" \text{ xsd:STRING.} \\
\nu(n_e) \text{ ldp:hasMembershipRelation } \text{dct:hasPart.} \\
\nu(n_e) \text{ dct:hasPart } \nu(n_c).} \quad \mathcal{R}_1
\]

\(^3\)https://www.w3.org/TR/cooluris/
① Each entity instance \( e = (n_e, C_e) \) is mapped to a \( \text{ldp:BasicContainer} \) with IRI \( \nu(n_e) \). This entity container maintains a membership triple \((\nu(n_e), \text{dct:hasPart}, \nu(n_a))\) for each component instance \((n_c, p_c, A_{c,e})\) attached to \( e \).

\[
(n_a, v, t) \in A_{c,e} \\
\nu(n_a) \text{ rdf:type } \text{ldp:RDFResource} . \\
\nu(n_a) \text{ dct:identifier } "n_{a}" \text{ xsd:string} . \\
\nu(n_a) \text{ dct:isPartOf } \nu(n_e) . \\
\nu(n_a) \text{ rdf:value } "\mu(v)^{\prime\prime}\nu(t)" .
\]

② Each component instance \( c = (n_c, p_c, A_{c,e}) \) is mapped to a \( \text{ldp:BasicContainer} \) with IRI \( \nu(n_c) \). This component container uses \( \text{dct:isPartOf} \) to indicate its containing entity container \( \nu(n_e) \) and maintains a membership triple \((\nu(n_c), \text{dct:hasPart}, \nu(n_a))\) for each attribute instance \((n_a, v, t)\) attached to \( c \). In addition, we use \( \text{rdfs:isDefinedBy} \) to indicate an authoritative resource \( \rho(p_c) \) semantically defining the component container \( \nu(n_c) \). We detail on \( \rho(p_c) \) in the next section.

③ Each attribute instance \((n_a, v, t) \in A_{c,e}\) is represented by a \( \text{ldp:RDFResource} \) with IRI \( \nu(n_a) \). This attribute resource uses \( \text{dct:isPartOf} \) to indicate its containing component container \( \nu(n_c) \). The triple \((\nu(n_a), \text{rdf:value}, "\mu(v)^{\prime\prime}\nu(t)"
\]

④ Since the RDF datatype abstraction is compatible with XML Schema, we rely on the data type support between an ECA runtime environment and XML Schema Types for datatype conversion. Given an attribute \((n_a, v, t) \in A_{c,e}\), we denote by \( \nu(t) \) the datatype IRI of the RDF-compatible XSD type corresponding to \( t \). The lexical form \( \mu(v) \) may be any lexical form, ie. a Unicode string in Normal Form C, from \( \nu(t) \)'s lexical space that represents the same value as \( v \). Extensions that handle domain-specific or user-defined datatypes beyond the RDF-compatible XSD types are expected to behave as outlined here.

V. AUGMENTING SEMANTICS

Rules ① to ④ provide a structural mapping from ECA runtime objects to Web resources described using the LDP vocabulary. Our structural mapping is generic and auto-generatable, but so far it only conveys little application-specific semantics.

A domain expert tasked with semantic augmentation thus requires support for specifying expressive RDF mappings that enable fine-grained term correspondences, literal transformations and structural graph transformations at dataset-level. Ideally, these RDF mappings should be dereferenceable and executable, self-contained and interoperably represented as RDF triples. Natural candidates for expressing and executing such RDF mappings are SPIN SPARQL\(^6\), RIF in RDF\(^7\), the LDIF framework\(^8\) or the R2R framework\(^9\).

In the scope of this paper and without loss of generality, we describe and publish such RDF mappings using the R2R Mapping Language [22]. Similar to SPARQL CONSTRUCT queries, a \texttt{r2r:Mapping} (cf. Figure 2(b)) has a \texttt{r2r:sourcePattern}, \texttt{r2r:targetPattern} and a \texttt{r2r:targetPattern}. The source pattern is matched against data generated from rules ① to ④ (cf. Figure 2(a)) and produces a set of variable bindings. Transformations define how variable bindings are transformed before being inserted into the target pattern. The target pattern is used to produce the triples resulting from the \texttt{r2r:Mapping} (cf. Figure 2(c)).

Rule ② uses \( \text{rdfs:isDefinedBy} \) to indicate an authoritative resource \( \rho(p_c) \) defining all instances of a component prototype \( p_c \in P_C \). Hence, a domain expert can publish her RDF mapping under \( \rho(p_c) \) and make it discoverable for Linked Data clients. By retrieving a representation of \( \rho(p_c) \), a Linked Data client will be instructed on how to locally render additional application-specific RDF triples. Note that execution of a RDF mapping may also be delegated to a suitable Linked Data Service (LIDS) [23]. We suggest \( \text{rdfs:seeAlso} \) (cf. Figure 2(a)) to indicate the respective LIDS invocation IRI.

VI. CONCLUSION

This paper presents a generic and auto-generatable structural mapping between Entity-Component-Attribute (ECA)

\(^6\)https://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/
\(^7\)https://www.w3.org/TR/rif-in-rdf/
\(^8\)http://ldif.wbsg.de/
\(^9\)http://wifo5-informatik.uni-mannheim.de/bizer/r2r/
runtimes and the W3C Linked Data Platform. Building upon this basic level of structural interoperability, we explain how domain experts may declaratively specify and publish expressive RDF mappings in order to convey the application-specific semantics of the respective ECA runtime objects. By executing the published RDF mappings, a Linked Data client is instructed on how to semantically interpret the dynamically exposed ECA runtime objects. A prototype implementation of the presented approach is available at https://github.com/tospie/eca2ld.

ACKNOWLEDGMENT

The work presented in this paper received funding from the European Union’s project FI-NEXT under grant agreement no. 732851, and by the Federal Ministry of Education and Research of Germany in the project Hybr-iT under support code 01IS16026A.

REFERENCES


