ECA2LD: From Entity-Component-Attribute runtimes to Linked Data applications

Torsten Spieldenner¹,², René Schubotz¹, and Michael Guldner¹

¹ German Research Center for Artificial Intelligence (DFKI)
² Saarbrücken Graduate School of Computer Science
firstname.lastname@dfki.de

Abstract. Large-scale IoT applications, like Smart Cities, are ever-changing pieces of software. Software built for such IoT environments needs a thorough design to be adaptable to the changes in the underlying systems. The Entity-Component-Attribute (ECA) pattern is well-suited for the design of changeable and maintainable software artifacts. However, the nature of large-scale IoT applications does not only enforce changeable, but also interoperable design of software components. For this, W3C working groups propose to use the Web as an IoT convergence platform. To unleash its full potential and to help to tackle pressing cross-domain interoperability issues, this emerging Web of Things is expected to evolve into a Semantic Web of Things which will heavily rely on Linked Data principles. While the generation of Linked Data from data storage layers has undergone thorough research, the Linked Data compliant exposure of dynamic run-time environments is to this day incomplete. Towards this end, we formalize the ECA design pattern and present an generic and auto-generatable mapping from ECA runtimes to a structure compliant with the W3C Linked Data Platform. This structural mapping may be declaratively augmented by domain-specific semantics, and lifts a software design pattern highly suitable for large-scale IoT applications to Semantic Web of Things.

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

1 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 [17, 14]. Especially in perdurable and large scale Internet of Things (IoT) platforms, software design with the intention to optimize changeability is imperative [27]. Towards this end, the architectural pattern of Entity-Component-Attribute (ECA) based software design is particularly well-suited [26]. 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 [19], however, the enablement of *seamless cross-domain interoperability* between independently developed IoT applications and platforms, one of the central challenges facing IoT [18], is not directly addressed by this design paradigm.

In this respect, the W3C Web of Things Working Group\(^3\) proposes to use the Web as an IoT convergence platform. The group develops initial standards for this *Web of Things* (WoT) [2] 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.

The *Semantic Web of Things* (SWoT) [21] will heavily rely on Linked Data principles [11] to semantically describe IoT entities in terms of their actions, properties, events and metadata [22] independent of the underlying IoT technology stacks. 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 IoT application data layouts to RDF
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 [6, 3, 16, 8, 15], little research has been conducted on the dynamic mapping of runtime environments [20, 12] to RDF. In particular, we are not aware of any works on how to leverage ECA-driven software applications on the Semantic WoT.

**Contributions.** 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 Platform\(^4\). Next, we explain how domain experts can declaratively augment these generated structural mappings with domain-specific semantics. Finally, we outline how a Linked Data client may materialize these application-specific RDF triples either locally or by delegation to a suitable Linked Data Service.

**Structure.** In the remainder of the paper, we first discuss current research in mapping approaches from non-RDF sources to RDF data. We then present a formalization of the Entity-Component-Attribute model as basis for changeable software in Section 3. In Section 4, we briefly outline the W3C Linked Data

\(^3\) https://www.w3.org/WoT/WG/

\(^4\) https://www.w3.org/TR/ldp/
Platform standard, and present our automated mapping for structural interoperability between ECA data and the Linked Data Platform in Section 5. Section 6 outlines a declarative augmentation of the automated structural mapping with semantics from domain-specific vocabularies. We conclude with summary in Section 7.

2 Related Work

The available literature investigating (semi-) automated data mappings from various data structures and serializations into the RDF data model is vast. However, little research has been conducted on the dynamic mapping of runtime environments [20, 12] to RDF. In what follows, we briefly survey the related literature.

RDBMS. Numerous work investigates how to translate between RDF datagraphs and relational databases. The W3C specifies a Direct Mapping (DM) from relational database structures directly to RDF. R2RML is a similar approach that enables customization of the mapping. Automatic procedures have been proposed to create the R2RML mapping, which yields RDF graphs similar to the results of Direct Mapping [8]. Bizer et al. [6] present D2RQ, an approach that translates semantic queries into native queries against non-RDF databases and maps the result to RDF.

OOP. Focusing on business logic rather than data storage layers, other work also investigated mappings between RDF data sources, and object-oriented programming (OOP) languages. Bartolos et al. [4] discuss mappings between object-oriented classes, and present a possibility to automatically create a class model from ontologies. The resulting class model then operates as access to the underlying RDF data. ActiveRDF by Oren et al. [20] provides an object-oriented API for scripting languages to operate on RDF datasets. However, they found that concepts of object-oriented programming are at times to strict to allow for an automated mapping. Limiting concepts are for example class inheritance rules, encapsulation, or class attribution. Hillairet et al. [12] show how to match the at this time widely used Eclipse Modeling Framework (EMF) with RDF data sources. Approaches that operate directly on OOP concepts struggle with OOP concepts like class inheritance, encapsulation.

Semi-structured Data. A third class of RDF mapping approaches uses as source semi-structured data, such as XML files, or comma separated values (CSV). Apache Any23, offered as library, Web service, or command line tool, translates a variety of source formats, such as CSV and YAML, to RDF representations in Turtle, Notation 3, and others. Dimou et al. [9] present RML, an extension to the R2RML W3C standard, to map between heterogeneous semi-structured

---

5 https://www.w3.org/TR/rdb-direct-mapping/
6 https://www.w3.org/TR/r2rml/
7 https://any23.apache.org/
data and RDF. Gupta et al. [10] provide Karma, a semi-automatic RDF extraction framework not only from relational database, but also from sources given as CSV, XML, or JSON.

Despite its wide application in different domains, and work that investigates how application design profits from semantic information inherent to ECA-based software, there exists to our knowledge no particular analysis about how to leverage Entity-Component-Attribute driven applications to Linked Data.

3 Changeability by Entity-Component-Attribute designs

In the following, we will give an outline of requirements we see for the design of changeable large-scale software projects. We give a formal definition of the established Entity-Component-Attribute model and discuss how it is suitable to fulfill the requirements towards changeable software.

3.1 Changeable software requirements

We derive requirements towards changeable software from the notion of aspect-oriented software design as presented by Kiczales et. al [13]. For this, it is necessary to avoid cross-cutting concerns in the code. As main pitfalls that break changeability of software, aspect-oriented design distinguishes:

**Code scattering**: Code that implements a concept or logic is distributed over several modules or classes. As a result, adding, changing, or removing logic from an application requires to change several modules at once. Code scattering is avoided by modeling software and its data in distinct modules for every task.

**Code tangling**: While code that implements a certain feature may be entirely contained in its own module, dependencies between modules can still break changeability of software. This happens for example if code of one module refers to, or makes calls to, code in another module. If one module changes its interface to which other modules make calls, all other modules need to be changed as well. A way to avoid code tangling is a data-centralistic approach by which separate software modules operate on a shared data layer, without direct calls between the modules.

3.2 Entity-Attribute Models

Above requirements are met by Entity-Attribute based software design. The understanding of Entity-Attribute models varies in literature. In the following, we summarise available variants and formalize the notation of Entity-Attribute models.
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 IoT Context Broker by Moltchanov et. al [19] keeps to the levels of entities and attributes.

The systems RealXtend [1, 7] and FiVES [25] include the notion of components (Entity-Component-Attribute pattern, ECA; see also Fig. 1). Components can be considered as prototypes of attribute sets that belong to the same concept. When a component is attached to an entity instance, a new instance of the component and the respective attribute set is created from this prototype. RealXtend and FiVES share this design with game engines like Unity3D\(^8\) and Unreal Engine\(^9\).

RealXtend equips components with application logic that is directly contained as code in the component implementation. Same holds for game engines.

The work by Wiebusch et al. [26] as well as the FiVES server system consider the Entity-Component-Attribute model as data model only. Logic is implemented in independent systems (referred to as plugins in FiVES), with a careful design of how external logic accesses the data.

We adapt the understanding of components as by Wiebusch, and FiVES, with logic implemented separately to avoid the tight coupling between components and their specific implementation as in RealXtend or game engines. We derive from this the following formal definition of the ECA architectural pattern.

---

\(^{8}\) http://www.unity3d.com

\(^{9}\) http://www.unrealengine.com
Let \( e \) denote an entity instance, and \( P_C \) denote the set of all component prototypes. Then we define the following sets:

- \( E \) is the set of all entity instances. An entity instance is defined as \( e = (n_e, C_e) \), with \( n_e \in \Sigma^+ \) being the unique identifier for \( e \) over alphabet \( \Sigma \), and \( C_e \) being the set of component instances attached to \( e \).
- \( C_e \) is the set of all component instances attached to an entity instance \( e \). A component instance is defined as \( c = (n_c, p_c, A_{c,e}) \), with \( n_c \in \Sigma^+ \) being the unique identifier for \( c \), \( p_c \in P_C \) being the prototype that \( c \) is an instance of, and \( A_{c,e} \) being the set of all attribute instances attached to \( c \).
- \( A_{c,e} \) the set of all attribute instances attached to a component instance \( c \). An attribute instance is defined as \( a = (n_a, v, t) \), with \( n_a \in \Sigma^+ \) being the unique identifier for \( a \), \( v \) denoting the attribute instance’s current value, and \( t \) denoting the ECA runtime type of \( a \).

The role of an entity within the application is by this entirely determined by the set of attached components. Components are implemented and operate independent of each other. This avoids the issue of code scattering. The \textit{composition-over-inheritance} principle of the model also avoids code-tangling, as it eliminates class inheritance and attribution hierarchies.

## 4 The Linked Data Platform

The W3C \textit{Linked Data Platform} recommendation provides best practices for read-write Linked Data applications on the Web. It describes how to model applications in terms of a minimal set of RDF resources (cf. Figure 2). Moreover access patterns to these different resources are specified for Linked Data clients.  

---

\(^{10}\) Image taken from https://www.w3.org/TR/ldp/#fig-ldpc-types
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. From the ldp:Resource are derived a number of resource types with the following roles:

ldp:RDFSource exposes general RDF data. Upon a HTTP GET request, it MUST return a full RDF graph in text/turtle format (or application/ld+json, if requested). HTTP PUT or HTTP PATCH can be used to update triples in the graph that is provided by the resource.

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 (container-uri, ldp:contains, document-uri). The ldp:BasicContainer does not require to specifically state the semantic relationship between the container resource itself, and its containing elements.

The W3C LDP recommendation specifies more concepts. In the scope of this paper, however, we will only make use of above concepts.

5 Structural Interoperability with ECA-based Systems

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. ① to ④), structural interoperability [23] between ECA runtimes and LDP servers is established.

We assume the existence of functions \( \nu : \Sigma^+ \to \text{IRI} \) and \( \rho : \mathcal{P}_\mathcal{C} \to \text{IRI} \) for minting fresh IRIs from identifiers and component prototypes. Although several guidelines exist for minting IRIs\(^{11}\), we do not make assumptions on \( \nu \) or \( \rho \).

\[
\begin{align*}
(n_e, C_e) & \in E \quad \forall (n_c, p_c, A_{c,e}) \in C_e \\
\nu(n_e) & \text{ rdf:type ldp:BasicContainer .} \\
\nu(n_e) & \text{ dct:identifier "n_e"^^xsd:String .} \\
\nu(n_e) & \text{ ldp:hasMemberRelation dct:hasPart .} \\
\nu(n_e) & \text{ dct:hasPart } \nu(n_e) .
\end{align*}
\]

① Each entity instance \( e = (n_e, C_e) \) is mapped to a ldp:BasicContainer with IRI \( \nu(n_e) \). This entity container maintains a membership triple \( (\nu(n_e), \)
dct:hasPart, ν(nc)) for each component instance (nc, pc, Ac,e) attached to e.

\[(nc, pc, Ac,e) \in C_e \quad \forall (na, v, t) \in A_{c,e} \]

\[
\nu(nc) rdf:type ldp:BasicContainer . \\
\nu(nc) dct:identifier “nc”^xsd:String . \\
\nu(nc) dct:isPartOf ν(ne) . \\
\nu(nc) ldp:hasMemberRelation dct:hasPart . \\
\nu(nc) dct:hasPart ν(na). \\
\nu(nc) rdfs:isDefinedBy ρ(pc). \\
\]

**(2)** Each component instance \(c = (nc, pc, Ac,e)\) is mapped to a ldp:BasicContainer with IRI \(ν(nc)\). This component container uses dct:isPartOf to indicate its containing entity container \(ν(ne)\) and maintains a membership triple \((ν(nc), dct:hasPart, ν(na))\) for each attribute instance \((na, v, t)\) attached to \(c\). In addition, we use rdfs:isDefinedBy to indicate an authoritative resource \(ρ(pc)\) semantically defining the component container \(ν(nc)\). We detail on \(ρ(pc)\) in the next section.

\[(na, v, t) \in A_{c,e} \]

\[
\nu(na) rdf:type ldp:RDFResource . \\
\nu(na) dct:identifier “na”^xsd:String . \\
\nu(na) dct:isPartOf ν(nc) . \\
\nu(na) rdf:value “μ(v)”^ν(t) . \\
\]

**(3)** Each attribute instance \((na, v, t) \in A_{c,e}\) is represented by a ldp:RDFResource with IRI \(ν(na)\). This attribute resource uses dct:isPartOf to indicate its containing component container \(ν(nc)\). The triple \((ν(na), rdf:value, “μ(v)”^ν(t))\) encodes the attribute’s current value \(v\) and type \(t\) as a typed literal \("v"^ν(t)\) (cf. ④).

**(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 \((na, v, t) \in A_{c,e}\), we denote by \(ν(t)\) the datatype IRI of the RDF-compatible XSD type corresponding to \(t\). The lexical form \(μ(v)\) may be any lexical form, ie. a Unicode string in Normal Form C, from \(ν(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.
Augmenting Domain-specific Semantics

The ECA model induces an implicit semantic understanding of the underlying data. The prototype of a component assigns to a component a specific concept that is modeled by that component. For example, Figure 1 defines a component prototype that describes a location in geo-coordinates.

Rules R1 to R4 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 does not express said application-specific semantics that are contained in the ECA model.

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 dereferencable and executable, self-contained and interoperably represented as RDF triples. Natural candidates for expressing and executing such RDF mappings are SPIN SPARQL\textsuperscript{12}, RIF in RDF\textsuperscript{13}, the LDIF framework\textsuperscript{14} or the R2R framework\textsuperscript{15}.

In the scope of this paper and without loss of generality, we describe and publish such RDF mappings using the R2R Mapping Language\textsuperscript{5}. Similar to SPARQL CONSTRUCT queries, a \texttt{r2r:Mapping} (cf. Figure 3(b)) has a \texttt{r2r:sourcePattern}, \texttt{r2r:transformation} and a \texttt{r2r:targetPattern}.

\textsuperscript{12} https://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/
\textsuperscript{13} https://www.w3.org/TR/rif-in-rdf/
\textsuperscript{14} http://ldif.wbsg.de/
\textsuperscript{15} http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/
The source pattern is matched against data generated from rules \( \oplus \) to \( \ominus \) (cf. Figure 3(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 3(c)).

Rule \( \circ \) uses \texttt{rdfs:isDefinedBy} to indicate an authoritative resource \( \rho(p_c) \) defining all instances of a component prototype \( p_c \in \mathcal{P}_C \). 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) [24]. We suggest \texttt{owl:sameAs} (cf. Figure 3(a)) to indicate the respective LIDS invocation IRI.

7 Conclusion

This paper presents a generic and auto-generatable structural mapping between Entity-Component-Attribute (ECA) runtimes and the W3C Linked Data Platform. First, we discuss the Entity-Component-Attribute model as suitable choice for changeable software, followed by a formal definition of the ECA design pattern. From this, a generic and auto-generatable structural mapping between ECA runtimes and the W3C Linked Data Platform is provided. 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.

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


