* A typical and pragmatic solution is to resort to Web Services and XML data, sharing schemas and namespaces, or to RESTful APIs, which are simpler to use and require that schemas (media types) be standardized or pre-agreed. In these technologies, both customer and provider are forced to implement full interoperability (for example, sharing a XML schema), even if only a fraction of the possible interactions is used. This leads to a greater coupling than needed.
* Interoperability is possible only if all these levels contribute to it. For example, if the intended effect is expressed correctly at the upper levels but one service sends a message in XML and the other is expecting JSON, they will not be able to interact. The same happens if services use exactly the same technologies and tools, but one expects an effect (such as receiving a payment) that the other does not provide.
* We use JSON to illustrate the details because it is easier to read than XML.

2

A typical and pragmatic solution is to resort to Web Services and XML data, sharing schemas and namespaces.

* The data in messages need to be serialized to be sent over the channel, using formats such as XML or JSON;

3

Data description languages, such as XML and JSON, merely describe data and their structure. If we want to describe services, we can use WSDL (a set of conventions in XML usage), but the resulting verbosity and complexity has progressively turned away developers in favour of something much simpler, namely REST. If we want a programming language suitable for distributed environments we can use BPEL [38], but again with an unwieldy XML-based syntax that forces programmers to use visual programming tools that generate BPEL and increase the complexity stack. For example, a simple variable assignment, which in most programming languages would be represented as *x=y*, requires the following in BPEL:

JSON is much simpler than XML and, thanks to that, its popularity has been constantly increasing [39]. The evolution of the dynamic nature of the Web, as shown by JavaScript and HTML5 [14], hints that data description is not enough anymore and distributed programming is a basic requirement. In the IoT, with machine to machine interactions now much more frequent that human interaction, this is of even greater importance.

With respect to XML, JSON points the way in data representation simplicity, but it too has a compatibility load, in this case with JavaScript. Like XML, it is limited to data (no support for operations) and its schema is a regular JSON document with conventions that enable to describe the original JSON document. With both XML and JSON, a schema is a specification separate from the data (documents) it describes. A schema expresses the associated variability, i.e., the range of documents that are validated by that schema.

The basis of data interoperability with XML and JSON is schema sharing. Both the producer and consumer (reader) of a document should use the same schema, to ensure that any document produced (with that schema) can be read by the consumer. This means that both producer and consumer will be coupled by this schema.

This may be fine for document sharing, but not for services (in which the service description and the data sent as arguments are the documents), since typically a service provider should be able to receive requests from several consumers and each consumer must be able to invoke more than one provider.

Therefore, we use a different approach to represent resources, which can be described in the following way:

* Listing 19.1 is what the programmer sees, a text program in a distributed programming language, with a syntax similar to usual and familiar object-oriented programming languages, instead of a verbose and cumbersome syntax of a data description language such as XML.  However, this is not necessarily what is used by the computing platform. A compiler transforms Listing 19.1 into binary code, like a Java compiler transforms Java source into bytecodes. This binary code is then interpreted, which guarantees its universality, as long as the interpreter is implemented in interacting computing nodes.  The binary representation uses a modified version of the TLV format (Tag, Length and Value) used by ASN.1 [41]. This not only supports the direct integration of binary information but also facilitates parsing, since each resource, primitive or structured, can be stepped over in a breadth-first traversal, thanks to the *Length* field (the resource size in bytes).  Resources (including messages) can be represented in three levels:
  1. Binary, no variable names (components are referred to by their position index in the resource);
  2. Variable names, in a dictionary that maps names to position indices;
  3. Source text, when available. Typically, runtime messages are generated directly  in binary, without source text. This is used mostly in design-time resources.

Tekrar BAK

4

* The serialization formats used in the Web (e.g., XML, JSON) are text-based  and thus verbose and costly in communications. Technologies have been de- veloped to compress text-based documents [18], such as EXI (Efficient XML Interchange) and others [19]. However, these are compression technologies, which need text parsing after decompression. Recent native binary serialization formats, such as Protocol Buffers and Thrift [20], do not have this problem;
* SON is a very popular serial- ization format, as a simpler alternative to XML. JSON Schema is currently just an IETF draft [36], but is raising interest as a simpler alternative to XML Schema.
* XML Schema and JSON Schema share many of their goals and characteristics, as expressed by Table 3. The fact is that even JSON Schema can be too much for the IoT, since usually RESTful APIs [29] just specify JSON as the media type, as- suming that concrete component names have been agreed between client and serv- er. This provides little support for interface verification when designing services. We need a better way of specifying schema level interoperability, particularly in the context of the IoT. The requirements to do so are expressed by the *Wish list (SIL)* column.

Tekrar inele

If we want a programming language suitable for distributed environments we can use BPEL [38], but again with an unwieldy XML-based syntax that forces programmers to use visual programming tools that generate BPEL and increase the complexity stack. For example, a simple variable assignment, which in most programming languages would be represented as *x=y*, requires the following in BPEL:

<assign>

<copy>

<from variable="y" />

<to variable="x" />

</copy>

</assign>