Chapter 13 – Standards and Open Source: What Exists So You Don't Build It Twice

13.0 Before you start

Where we are. Chapter 12 evaluated platforms without naming one. This chapter has the opposite obligation: the book has already named six standards, promised each of them an explanation, and in one case said in as many words that two of them were "named once each and did not explain" (Chapter 7 Sec. 7.13). Those explanations are due.

The chapter's title comes from Chapter 1 Sec. 1.4, which put the case in one sentence: *every hour you spend re-inventing an interface that a standard already defines is cost with no value attached, and Chapter 13 is about not paying it twice.* That is a promise about **cost**, so this chapter answers it with a cost calculation rather than with a catalogue.

The register, unchanged from Chapters 9 to 12. You have read a specification before, and you know what it feels like to adopt one that nobody else in the room had adopted.

> **This chapter does not teach any standard. It teaches what a standard is worth, to whom, and under what condition – and the condition turns out to be countable.**

Three things the field says about itself, which shape everything below. It is worth putting them at the top rather than burying them, because they are not what a chapter about standards usually opens with:

- Adoption of ISO 23247, the manufacturing twin reference architecture, is "still in its embryonic stages", and "we are far from being able to properly measure the compliance of existing digital twin architectures" with it [1].
- Existing digital-twin description formats "present information in an incompatible way, hindering interoperability between different Digital Twins" [2].
- Independently developed open-source Asset Administration Shell stacks turn out to be **incompatible with one another** [3].

Three independent sources saying that the standards landscape does not yet reliably deliver the thing standards exist to deliver. That does not make standards worthless – several of them pay handsomely, and Sec. 13.3 says which. It means **the value has to be argued for a specific case rather than assumed**, which is what Sec. 13.4 does.

What you are assumed to know. Everything so far. Especially: Chapter 1's cost estimate in Sec. 1.8.7 and its "don't pay twice" claim; Chapter 2's two standards-derived definitions in Sec. 2.5; Chapter 3's ISO 23247 domains in Sec. 3.5.3; Chapter 6's model-exchange versus co-simulation split; Chapter 7's credibility argument and its risk-informed principle; Chapter 9's five transport properties; Chapter 10's context model and provenance chain; and **Chapter 12's cost model** – `B`, `F`, `r`, `a`, `p` – which this chapter reuses rather than replacing.

The maths budget. As Chapters 9 to 12: more arithmetic welcome, no new mathematics. One set piece, and it is Chapter 12's model with a different `F`.

What this chapter deliberately does not cover. Teaching any specification. A

current inventory of standards – the landscape survey this chapter draws on is a 2021 deliverable and is cited for its framing and method rather than its list, which Sec. 13.2 says explicitly. Ontology formalism – Chapter 10 excluded it with reasons recorded and this chapter does not reopen it. Legal exposition of regulation – Sec. 13.3.6 covers only what a regulation obliges you to *produce*. Data spaces, cross-organisation sharing and ecosystems – Chapter 15, and the counterparty rule is the bridge. Security standards in depth – Chapter 3 Sec. 3.3.3. Product selection – Chapter 12. Licence law.

Learning objectives

By the end of this chapter, you will be able to:

1. **Classify** a standard into one of four kinds, and **determine** whether it is one you choose or one you are given.
2. **Explain** what each standard this book has named is for, what adopting it costs, and the one condition under which it pays.
3. **Compute** a standard's adoption using Chapter 12's cost model and **apply** the counterparty rule to decide.
4. **Distinguish** three different things called open source in a twin project, and **check** the four things that matter before depending on one.
5. **Name** the published twin exemplars you can read instead of guessing, and **state** what each one demonstrates.

13.1 What a standard is worth, and to whom

13.1.1 Chapter 1's promise, taken literally

Chapter 1's claim was economic: re-inventing a defined interface is cost with no value attached. True, and incomplete, because it invites a conclusion that does not follow – *therefore adopt the standard*. The missing term is what adoption itself costs.

Adopting a standard is not free and is frequently not cheap. You read the specification, you map your model onto its abstractions, you acquire or build tooling, and you deal with the parts of your system that do not fit its shape. Chapter 12 gave that quantity a name – `F`, the fixed integration cost – and observed that it is paid before anything has been supplied.

So Chapter 1's sentence, completed:

> Re-inventing a defined interface is cost with no value attached **when somebody else is already using that definition**. When nobody is, adopting it is also cost with no value attached, and it is usually the larger of the two.

13.1.2 What the three honest notes mean for you

Sec. 13.0 quoted three findings. Read as engineering advice rather than as criticism, they say:

Do not assume a standard is implemented consistently. Asset Administration Shell (AAS) implementations are incompatible with each other [3], which means "we both use AAS"

is not the same statement as "we interoperate". **Test the exchange, do not infer it from the name** – which is Chapter 12 Sec. 12.3.2's export test in a different guise.

Do not assume a reference architecture is adopted. ISO 23247 adoption is early and compliance is not yet measurable [1]. So citing it in a design document is a communication act, which is valuable, and not a compatibility claim, which it is often mistaken for.

Do not assume description formats compose. Different twin-description formats present information incompatibly [2], and work exists precisely to bridge them – which tells you the bridging is not free.

None of that is a reason to skip standards. It is the reason the rest of this chapter is a decision procedure rather than a recommendation.

13.1.3 The counterparty rule

Here is the idea the chapter is built on, and it is worth stating before any standard is named:

> **A standard's value is not what it saves you from writing. It is what it saves you from *agreeing*.**

Consider what you would do without one. You would define an interface, write it down, and give it to whoever needs to speak it. That is a **bilateral agreement**, and it works. Its costs are that it must be negotiated, documented, versioned, and re-negotiated with each new party – and that nobody outside your project has ever heard of it.

Figure 13.1 is why the value scales the way it does, and why "with nobody" is a real answer rather than a rhetorical one.

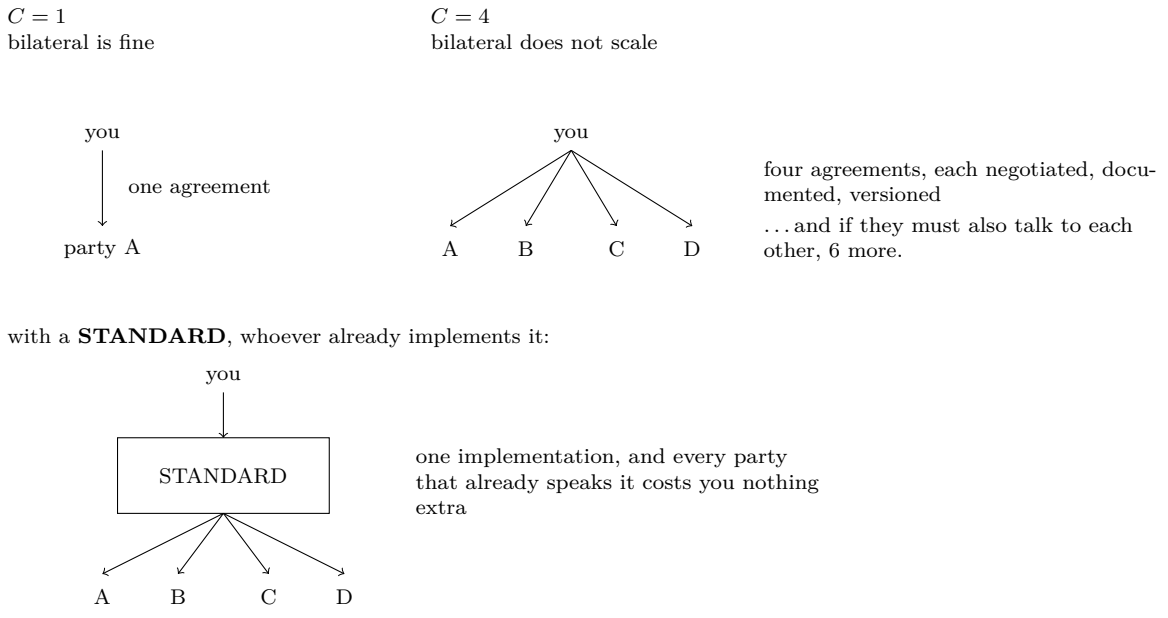


Figure 13.1 The counterparty rule. The quantity that decides is a head count of parties who already implement it – not the standard's quality, scope or maturity.

A standard is a bilateral agreement that somebody else already negotiated and that other people have already implemented. **So its value scales with how many people you need to agree with**, and with nobody, it has none.

> **The counterparty rule.** Count the parties you must exchange something with, who already implement the standard. Call it `C`. Adopt when
>
> `C x I > F`
>
> where `I` is what one bilateral integration would cost you and `F` is Chapter 12's fixed adoption cost.

Sec. 13.4.3 puts numbers on it. The shape of the answer, in advance:

Counterparties	Usually right
0 – you are the only party	**Skip.** Write your own interface and document it
1 or 2	**Usually a bilateral agreement**, unless the counterparty has already adopted, in which case adopting is free for you
3 or more	**Adopt**
Any number, but one of them is an auditor or a regulator	**The arithmetic does not apply** – see Sec. 13.2.2

A counterparty is not always another company. Your future self is not one. But a second team in your own organisation is, a supplier delivering models is, a customer receiving reports is, an auditor reading your evidence is, and – in Chapter 15's world – another twin is. Counting them honestly is the whole decision.

13.2 The four kinds, and the division that matters more

13.2.1 Four kinds of standard a twin meets

They are not interchangeable and they fail differently.

Kind	What it specifies	Example	Adopting it gets you
Vocabulary	Words, roles, a reference architecture. **No bytes**	ISO 23247	A shared conversation, and an audit trail that uses recognised terms
Interface	An exchange, precisely enough that two implementations interoperate	the Functional Mock-up Interface (FMI), OPC UA	Actual interoperability, and tooling you did not write

Kind	What it specifies	Example	Adopting it gets you
Process	What evidence you must produce and how	American Society of Mechanical Engineers (ASME) VV-40	A structure for Chapter 7's argument, and credibility with someone who knows the standard
Regulation	What you are obliged to do, with legal force	AI and sector-specific law	Permission to operate

The commonest category error in twin projects is treating a vocabulary standard as an interface standard. ISO 23247 will not let two systems exchange anything, because it specifies nothing to exchange. Chapter 3 Sec. 3.5.3 already said the honest version: it is a reference model pitched too abstractly for code to be generated straight from it [4]. That is not a flaw – vocabulary is genuinely useful – but a project that adopts it expecting interoperability has bought the wrong thing.

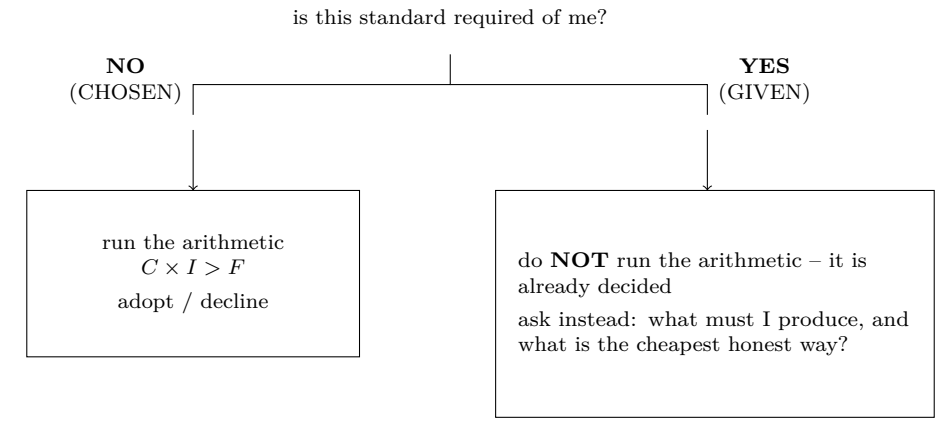
13.2.2 Chosen and given

More important than the four kinds:

> **A standard you *choose* is subject to Sec. 13.1.3's arithmetic. A standard you are *given* is not.**

If a customer's purchase order requires ISO 23247 vocabulary, if an auditor works to ASME VV-40, if a regulator's pathway assumes a particular evidence form, then the cost-benefit calculation has already been made by somebody else and your only question is **what does this oblige me to produce, and what is the cheapest honest way to produce it?**

Figure 13.2 puts the two paths side by side, because the expensive mistake is running the wrong one.



the two common errors, and they are mirror images:

evaluating a **GIVEN** standard → weeks spent deciding something already decided
skipping evaluation on a **CHOSEN** one → adopting a standard with $C = 0$

Figure 13.2 One question, asked first, routes the whole chapter. An hour here saves the weeks named below.

Two practical consequences.

Find out which you are in, first. An hour spent asking "is this required, or are we choosing it?" saves weeks. Teams routinely run an adoption evaluation on a standard they have no choice about, and routinely skip the evaluation on one they do.

A given standard still has a cheapest form. Chapter 7's credibility argument was five parts on two pages. If an auditor requires a particular structure, the work is mapping your five parts onto their structure – not producing different evidence. **Evidence you already have, presented in somebody else's shape, is the whole of most compliance work**, and recognising that is the difference between a two-week task and a two-month one.

13.2.3 A note on the landscape, and on landscape surveys

There is a genre of document that enumerates every standard relevant to digital twins. The most useful one in this book's corpus was produced for exactly the audience this chapter serves – it exists because implementing twins in small and medium enterprises "is much facilitated by deploying standards", and it identifies the relevant standardisation bodies and the standards each works on [5].

Read that sentence for its method and not for its list. That deliverable is from 2021, and any inventory of this landscape ages faster than the standards in it. What transfers is the approach: identify the bodies, find which standards touch your domain, and then – the step the surveys cannot do for you – **apply Sec. 13.1.3 to each one**.

13.3 The six this book has already named

Each subsection answers three questions: what is it, what does adopting it cost, and when does it pay.

13.3.1 ISO 23247 – vocabulary, and the one Chapter 3 owed you

What it is. A framework and reference architecture covering digital twins in the manufacturing domain. Chapter 2 Sec. 2.5.2 took its definition verbatim – a twin is a "fit-for-purpose digital representation of an observable manufacturing element, with a means to enable convergence between the element and its representation at an appropriate rate of synchronisation" [6] [7]. Chapter 3 Sec. 3.5.3 took its four domains: observable manufacturing, device communication, digital twin, and user, with the device communication domain splitting into data collection and device control sub-entities [8], [9]. It has been used as the frame for component catalogues [10], mapped onto by other frameworks [11], and applied to worked use cases [12], [13].

What it costs. Two to five engineer-days to read and map onto your architecture. **No tooling, because there is nothing to implement.**

When it pays. When you have a counterparty who uses its vocabulary – a manufacturing customer, an auditor, a partner in a consortium. Then it converts a translation problem into a shared one. When you do not, it buys a shared conversation with yourself.

What to know before citing it. Adoption is early: an analysis of 29 manufacturing twin architectures against the standard found several multinationals starting to use it while adoption remains "in its embryonic stages", and concluded that measuring compliance is not yet really possible [1]. **So "ISO 23247 compliant" is a weaker claim than it sounds, in both directions** – it is hard to verify and therefore hard to rely on.

And the gap Chapter 7 already found. ISO 23247 does not cover verification, validation and uncertainty quantification – the credibility assessment of Chapter 7 – and this is named as a hole in the standard by people working on it [8]. A twin that is ISO 23247 conformant has said nothing about whether it should be believed.

13.3.2 The Functional Mock-up Interface, and the one Chapter 6 owed you

What it is. A standard for packaging a model as a component another tool can execute, shipped as a Functional Mock-up Unit (FMU). Chapter 6 Sec. 6.8 taught the mechanism and the one distinction that matters – model exchange, where the importer's solver steps the model, versus co-simulation, where the model brings its own solver – and Chapter 6 said Chapter 13 owed the standard's own story.

What it costs. Modest and asymmetric. *Consuming* an FMU is nearly free – open-source and commercial importers exist, and lightweight frameworks for wiring FMI-based twins are published [14]. *Exporting* one from your own model is real work, and the tooling depends on how your model was built.

When it pays. The moment a model crosses an organisational boundary – which is why it exists. A supplier who delivers an FMU has delivered something you can run without knowing what tool produced it, and that is a genuine "do not build it twice". Practitioners surveyed about co-simulation ask for the same thing in different words: they want units that can be coupled without the coupling itself becoming a research problem, and they want the interface for doing so to be one somebody else already specified [15].

When it does not. When the model never leaves your team and evaluates in nanoseconds, which is the demonstrator's case (Chapter 4 Sec. 4.5.3). Wrapping a nine-line water-

balance model as an FMU is Chapter 1's unpaid work.

The honest caveat. FMI standardises the *interface*, not the model's quality. Chapter 7's entire apparatus still applies to an FMU you were handed, and Chapter 7 Sec. 7.7.5's contract clauses are how you ask for it.

13.3.3 OPC UA – interface, and the one Chapter 9 owed you

What it is. The industrial-automation communication standard, and the reason Chapter 9 Sec. 9.4.4 singled it out: it carries an **information model** as well as data. It answers "what is this value?" as well as "what is it now?", which is the distinction Chapter 10 spent a chapter on – context, not just measurement.

What it costs. Substantial if you are implementing a server; moderate if you are a client against somebody else's. The information-modelling half is where the real effort is, because it requires deciding what your assets *are* before you can describe them – which is Chapter 10 Sec. 10.4's work, done in somebody else's notation.

When it pays. When your physical twin already speaks it. This is the common case in industrial settings and it makes the decision trivial: the counterparty count is at least one and the alternative is a translation layer you own forever.

When it does not. When your source offers HTTP and nothing else, which is the demonstrator's situation. Chapter 9 Sec. 9.9.1 scored that plain REST API five out of five on the properties that actually matter, and adding OPC UA would mean building a server to talk to yourself.

The interoperability work around it is real and worth knowing exists [16], [17], including proposals for mapping between OPC UA and the Asset Administration Shell [18]. That such mappings are an active subject is itself the point of Sec. 13.1.2: two standards do not automatically compose.

13.3.4 The Asset Administration Shell – interface, and Chapter 10's context model in somebody else's notation

What it is. A specification for describing an asset in a machine-readable way: an administration shell containing submodels, each identified and referring to a submodel template that gives it meaning [19]. It is the Industry 4.0 world's answer to the question Chapter 10 Sec. 10.4 asked – how do you describe what a thing *is*, alongside what it is doing?

What it costs. The highest `F` of anything in this section. You need the specification, tooling, and a mapping from your context model onto submodels – and Chapter 10's validity intervals, which make context historical, are not evidently expressible. **Budget eight to fifteen engineer-days before anything works, and check the history question early**, because it is the one most likely to force a compromise.

When it pays. When you exchange asset descriptions with other organisations – a supply chain, a customer's plant system, a marketplace. The counterparty count is the whole argument and it is often genuinely above three.

The two cautions, both from the corpus. Open-source implementations are incompatible with each other [3], so choosing AAS does not finish the decision – you also

choose an implementation and inherit its interpretation. And description formats more broadly present information incompatibly [2], which is why bridging work exists. **Run Chapter 12 Sec. 12.3.2's export test between your implementation and your counterparty's before designing around the exchange.**

And a pointer rather than a lecture. The asset-model question is Chapter 10 Sec. 10.4.6's, and this book deliberately does not teach the formalisms. If your context is seven well-maintained tables with validity intervals, you are ahead of most, and AAS is a way to *export* that, not a replacement for having it.

13.3.5 ASME VV-40 – process, and the second one Chapter 7 owed you

What it is. A standard for assessing the credibility of computational models, from the medical-device world. Chapter 7 Sec. 7.1.2 named it as the origin of the chapter's organising idea – **risk-informed credibility**: the depth of the assessment is set by the consequence of being wrong, not by how much rigour the team enjoys [20].

What it costs. Reading it is a couple of days. Working to it is not a software cost at all: it is the cost of producing the evidence, which Chapter 7 already costed at a fortnight of discipline for an advisory twin.

When it pays. Whenever there is a reviewer – an auditor, a regulator, a customer's engineering function. Then it supplies a shape they already recognise, and Sec. 13.2.2's point applies: you are presenting evidence you should have anyway, in their structure.

When it does not. Never quite "does not" – **its idea is worth having even if you never cite it**, because Chapter 7's whole method is that idea. But formally conforming to it, for a twin nobody external will review, buys paperwork.

The limitation Chapter 8 found, restated here because it belongs to the standard rather than to the technique. VV-40 assumes a credibility assessment can be done once for a knowledge-driven model, and for data-driven models that assumption is explicitly unresolved [20]. Chapter 8 Sec. 8.6.1 built the cadence answer. **A standard that predates your model class is a starting structure, not a verdict.**

13.3.6 AI, data and regulation – the group Chapter 8 and Chapter 10 owed you

Three strands, and they behave differently from the five above because they are mostly *given*.

AI quality standards. Several standards aimed at machine-learning systems are still being written rather than published, and a survey of AI in cyber-physical systems tracks them as a group: some fix the vocabulary, some say what "good enough" has to mean for a model's performance, some govern the data it was trained on, and some prescribe how any of that gets measured [21]. What matters for you is the direction: **the things Chapter 8 asked you to do voluntarily – characterise the training data, state the coverage, report the two rates – are becoming things you will be asked to evidence.** Building them in now is cheaper than retrofitting.

Regulation. The compliance landscape for AI is real and moving [22], and in regulated sectors the pathway is the subject rather than a detail – healthcare twin work treats the

credibility assessment as the hardest part of getting an in-silico method accepted [20], [23]. **This chapter's position is narrow and deliberate:** a regulation is a given standard, so the only question is what it obliges you to produce, and the answer is almost always a form of Chapter 7's credibility argument plus Chapter 10's provenance. If you have those, compliance is presentation. If you do not, no standard will help.

Research-data practice. Chapter 10 deferred the Findable, Accessible, Interoperable and Reusable (FAIR) principles here. Its content – that data should be findable, accessible, interoperable and reusable – is not a twin standard and is closer to a discipline. **For a twin, the useful translation is Chapter 10 Sec. 10.7's reproducible dataset**: a named, re-runnable definition with a frozen watermark is the operative form of "reusable", and it is worth more than a compliance checkbox.

Security. Named once, per Chapter 3 Sec. 3.3.3 and Chapter 9 Sec. 9.8.4. Twin threat classifications organise by functional layer [24], and the standards that apply are your sector's rather than the twin field's. This chapter does not extend them.

13.3.7 The six, on one page

Standard	Kind	F, engineer-days	Pays when
ISO 23247	Vocabulary	2-5	A counterparty uses its words
FMI	Interface	1 to consume, 5-10 to export	A model crosses an organisational boundary
OPC UA	Interface	5 as client, 15+ as server	Your physical twin already speaks it
Asset Administration Shell	Interface	8-15	You exchange asset descriptions with other organisations
ASME VV-40	Process	2 to read	Anybody external will review your evidence
AI / data / regulation	Mostly given	Varies	You are told

Figure 13.3 attaches the same six to Chapter 3's anatomy, which is the view that answers "do I need this one?" faster than the table does – a standard you have no component for is a standard you have no counterparty for.

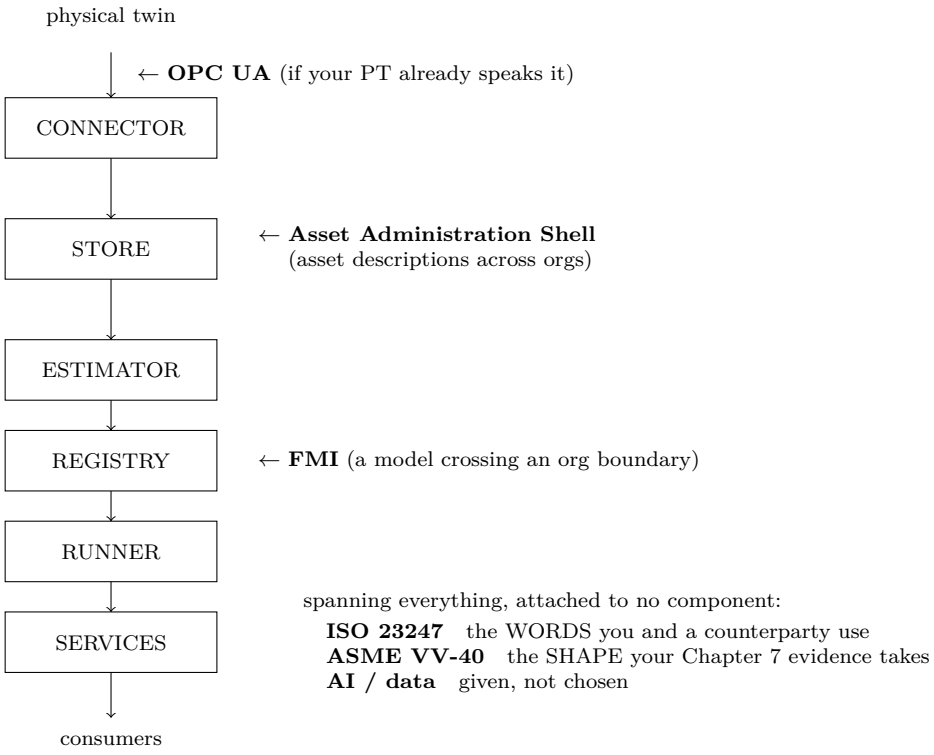


Figure 13.3 Where each of the six attaches. Four bind to a component and answer "do we exchange this, with whom?"; two bind to the conversation instead.

The `F` figures are illustrative, in the sense Chapter 9 Sec. 9.0 used for datasheet numbers: the method transfers, the values depend on your team and your tooling, and they are measurable by spending a day on a prototype. What is *not* illustrative is their ordering, which is stable: vocabulary is cheap, interfaces are moderate, asset description is expensive, process is cheap to read and expensive to satisfy.

13.4 The adoption calculation

13.4.1 A standard is a platform with a different `F`

Chapter 12's model needs no modification:

```
F  = fixed cost of adopting: reading, mapping, tooling, and the misfits
r  = fraction of the build it removes
B  = engineer-days to build the twin
```

What is different is where `r` comes from. A platform's `r` is code you do not write. A standard's `r` is mostly *agreement you do not negotiate*, which is why Sec. 13.1.3 counts counterparties instead of components.

13.4.2 The demonstrator's audit, against Chapter 1's four lines

Chapter 1 Sec. 1.8.7 split the build into four lines of 1.5 weeks each. Run every standard in Sec. 13.3.7 against them:

Build line	7.5 days each	Which standard could remove any of it?
Data path: poll the REST API, land readings	7.5	**None.** The source speaks HTTP. OPC UA would mean building a server to talk to ourselves
Water-balance model and parameter fitting	7.5	**None.** FMI packages a model; it does not write one
Residuals, thresholds, alerting	7.5	**None.** No standard specifies Chapter 11's three outcomes or Chapter 7's threshold
Deployment, documentation, handover	7.5	**Marginally.** ISO 23247 vocabulary in the documentation, perhaps a day saved arguing about names

```
total r across all six standards   = about 1 engineer-day
total F if all six were adopted    = 25 to 50 engineer-days
```

The audit's answer is unambiguous and slightly uncomfortable: for this twin, essentially no standard removes any build. Chapter 1's "don't pay twice" was a true statement about a situation the demonstrator is not in.

13.4.3 The counterparty count, computed

Sec. 13.1.3's rule with numbers. Take a standard with `F` = 10 engineer-days – an interface standard, mid-range – and suppose one bilateral integration costs `I` = 4 engineer-days to define, document and maintain.

```
adopt when  C x I > F
            C x 4 > 10
                C > 2.5
```

Three counterparties. Below that, define your own interface and write it down; above it, adopt.

Now count the demonstrator's counterparties, honestly:

Candidate	A counterparty?
The physical twin's REST API	**No** – but not because the question does not arise. Chapter 1 Sec. 1.8.1 noted that the demonstrator's own documentation names a *Remote Connecter*, "either a person or a computer running a Digital Twin": its authors designed for a party at the other end. `C` is zero because **they already supplied the interface**, which is the counterparty rule working rather than an exception to it
The lab's own researchers	No. They read a plot
A future second bench	No – **a future party is not a counterparty**, and Chapter 12 Sec. 12.8.4 already made "a second twin is funded" an expiry trigger
An external auditor	None exists

```
C = 0.  C x I = 0.  Adopt nothing.
```

And the inversion, which is the same arithmetic pointed at a different project. A manufacturing supplier delivering twin-ready components to five customers, each of whom would otherwise need a bespoke integration:

```
C = 5,  I = 4,  C x I = 20  >  F = 10.   Adopt, comfortably.
```

> **Same standard, same cost, opposite decisions, and the only variable is `C`.** That is the chapter's central claim and it is why a research bench and a manufacturing supply chain reach different answers without either being wrong.

13.4.4 Four ways the count is got wrong

Counting parties who do not implement it. A customer who has *heard of* AAS is not a counterparty for AAS purposes. The question is whether adopting lets you skip an integration, and it only does if they have already adopted.

Counting your future self. "We will want this later" is Chapter 12's expiry-trigger pattern, not a counterparty. Write the trigger; do not pay now.

Not counting internal parties. A second team in your own organisation building a second twin *is* a counterparty, and this is the count most often missed – because it feels like something you can settle in a corridor. You can, once. Chapter 12 Sec. 12.6.3's third failure is what happens after that.

Not counting the auditor. If anyone external will read your evidence, Sec. 13.2.2 says you are in the given case and the arithmetic does not apply.

13.4.5 What a standard buys that the arithmetic misses

Three things, and they are the reason the answer is not always the one the calculation gives.

An export format that outlives you. Chapter 12 Sec. 12.3.2's exit question – can you get your data out in a form another system can consume? – has a much better answer when there is an agreed format to export *into*. A twin whose data exports to a standard shape has a cheaper worst day, and that value is real and does not appear in `r`.

Recognition. Chapter 7's credibility argument is more persuasive when its structure is one the reader has seen before. This is the whole value of a process standard and most of the value of a vocabulary one.

A decision somebody else made carefully. A standard encodes choices – what to name things, what to make mandatory, how to version – that you would otherwise make in an afternoon. Even where you do not adopt it, **reading one is a cheap way to find the questions you have not asked**, and Sec. 13.3.4's history question is an example: reading the asset-description specification is what prompts you to check whether Chapter 10's validity intervals are expressible at all, which is worth having checked whether or not you adopt anything.

13.5 Open source: what exists so you do not build it twice

The chapter's second half, and in practice the one that pays for the demonstrator.

13.5.1 Three different things called open source

They have different risks and get conflated constantly.

Kind	Example in this book	What you depend on	Risk if it goes away
Ordinary components	A time-series store, a broker, a plotting library, a scheduler	A mature project with many users outside your field	Low. Alternatives exist and interfaces are similar
Twin frameworks	Purpose-built twin platforms and frameworks [25]–[28]	A smaller community, often one research group or consortium	Moderate to high. Chapter 12's `F` was mostly about these
Exemplars	Published, documented, runnable twins (Sec. 13.5.3)	Nothing at runtime – you read them	None. You copied an idea

The third row is the one people miss, and it is the safest dependency in the table, because it is not a dependency. Chapter 12's whole analysis was about the first two.

13.5.2 Four checks before depending on one

Not a licence lecture. Four things, in the order they bite.

1. Does it do what you need, tested? Chapter 12 Sec. 12.3.2's export test, applied to a library: run your actual data through it before designing around it.

2. Who maintains it, and what happens if they stop? For an ordinary component, "many organisations" is the answer you want. For a twin framework, it is frequently one research group with a grant. **That is not a reason to avoid it** – it is a reason to know it, to keep Chapter 12 Sec. 12.6.2's seams intact, and to be able to say what forking would cost.

3. Does the licence say what you think? Read it, once, and check it against what you will do with the output – especially if you will ship the twin to a customer or embed it in a product.

4. Can you contribute? Not altruism. Sec. 13.5.4.

13.5.3 The exemplars: twins you can read instead of guessing

This is the most concrete answer this chapter has to its own title. A handful of digital twins are published with their code, their data and their write-ups. Reading one is worth more than reading three surveys, and each of these has appeared earlier in this book.

The incubator. A physical incubator with heaters, fans and temperature sensors, twinned end to end. It has: a tutorial-length treatment of the whole system, its services and its architecture [29]; a case-study chapter that shows the calibration done properly, including a two-parameter model compared against a four-parameter one on real data [30]; and – the unusual part – **a survey that implements the same case study across five open-source twin frameworks** [25]. That last one is the closest thing the field has to a controlled comparison, and Chapter 12 Sec. 12.13 already recommended it for that reason.

What it demonstrates: Chapter 7's calibration on real hardware, Chapter 11's service set, and Chapter 12's framework question, all on one system you can run.

GreenhouseDT. A greenhouse exemplar built around a knowledge graph that records how the physical system is structured *and how that structure changes*, with plants that move between shelves [31]. This book has cited it since Chapter 1 for binding, and Chapter 10 Sec. 10.4 for context history. Related work takes the same system further into semantic self-adaptation [32].

What it demonstrates: Chapter 3's binding problem and Chapter 10's context history, in the case where the physical configuration genuinely changes.

The plant-controller demonstrator. This book's own. The physical twin exists, it is documented, and its firmware and API are published – which is why Chapter 1 Sec. 1.8.1 could describe the hardware from the demonstrator's own documentation and source rather than from an idealisation.

What it demonstrates: what a physical twin looks like *before* anyone twins it – which is the state most readers will actually meet, and the one exemplars usually skip.

Why this section is the chapter's best answer to "don't build it twice". A standard saves you an agreement. A framework saves you code. An exemplar saves you the six weeks you would spend discovering that calibration needs two experiments, that context needs history, and that a detector needs three outcomes. **The book you are reading is one long argument that those discoveries are expensive; the exemplars are where you can watch somebody else make them.**

13.5.4 Contributing back is cheaper than forking

The engineering case, not the ethical one.

You will need a change. The options are to fork, to carry a patch, or to contribute upstream. Forking means you own it forever, including every upstream fix you now have to merge by hand. Carrying a patch is a fork with optimism. Contributing upstream costs a review cycle and then costs nothing ever again.

Two twin-specific notes.

Twin frameworks are small enough that contributing works. Unlike a major infrastructure project, a framework maintained by one research group will usually engage with a well-made contribution quickly.

What you contribute is often not code. Chapter 12 Sec. 12.3.2's export test, run against a framework and written up, is a contribution. So is a worked example on different hardware – which is, exactly, another exemplar.

13.6 Worked example: the demonstrator's standards decision

13.6.1 The counterparty count

$\texttt{C} = 0$, per Sec. 13.4.3. One project, one physical twin, one team, one audience of researchers reading a plot. No auditor, no regulator, no supplier, no second team.

13.6.2 The decision, standard by standard

Standard	Decision	Reason
ISO 23247	**Borrow the vocabulary, do not claim conformance**	Its words are useful in the documentation and cost nothing. "Conformant" is unverifiable [1] and would be a claim we cannot support
FMI	**No**	The model is nine lines and never leaves the team. Revisit if a modelling group is contracted

Standard	Decision	Reason
OPC UA	**No**	The source speaks HTTP, five out of five on Chapter 9's properties
AAS	**No**	Highest `F` in the table, zero counterparties, and Chapter 10's validity intervals may not map
ASME VV-40	**Use the idea, skip the conformance**	Chapter 7's whole method already is its idea. No external reviewer exists
AI / data / regulation	**Not applicable**	No learned model in production yet; Chapter 8's proposal 2 would change this

Total adopted: one vocabulary, borrowed. And that is the correct answer for this twin, arrived at by counting rather than by taste.

13.6.3 What is adopted instead

Sec. 13.5's first row does the work the standards could not – and the way it does it is worth getting right, because the obvious accounting is wrong.

The tempting version. Credit ordinary open-source components with removing days from the 30: three for the store, two for the plot, one and a half for scheduling and retries. Six and a half days for about one day of adoption cost, against the standards' one day for twenty-five to fifty.

Why that is wrong, and the way it is wrong is the point. Look again at Chapter 1's first line: *"poll the REST API,* ***land readings in a time-series store*** *– 1.5 wk"*. Seven and a half days is what it costs to **wire up** a time-series store. It is not remotely what it costs to write one. The same is true of every other line: the model line assumes a numerical library, the alerting line assumes a scheduler, the deployment line assumes a container runtime.

> **`B` = 30 already presupposes ordinary open-source components. They are not a saving against it – they are the reason it is 30.**

So the right question is what `B` would be without them. Writing a time-series store, a plotting library, an HTTP client and a scheduler is not a fortnight's work in any universe; it is hundreds of engineer-days, and **nobody on the project ever considered it, which is exactly why it appears nowhere in the estimate.**

Kind of re-use	Visible in the estimate?	What it is worth
Twin-specific standards	**Yes** – a line somebody must justify	About 1 day removed, 25 to 50 spent
Ordinary open-source components	**No**	More than the entire project

Kind of re-use	Visible in the estimate?	What it is worth
Exemplars (Sec. 13.5.3)	**No**	The weeks you would spend rediscovering Chapters 7 to 11

That table is the chapter's real finding, and the middle column is where it lives. The re-use that pays most is invisible in every estimate, because an estimate only counts what somebody considered building. The re-use that is *visible* – standards, with their specifications, their conformance claims and their line in the budget – is the one that has to justify itself, and for a single-organisation twin it usually cannot.

Which is not an argument against standards. It is an argument against judging them by the yardstick that flatters components. Standards are paid for by counterparties, not by lines of code, and Sec. 13.4.3 is where they are judged properly.

13.6.4 And the exemplars, which cost nothing

Before writing anything, read the incubator's tutorial and case study [29], [30] and GreenhouseDT [31]. **Zero adoption cost, no dependency, and they demonstrate three things this book spent chapters on.** If the standards audit is the chapter's negative result, this is its positive one.

13.6.5 The expiry triggers

Following Chapter 12 Sec. 12.8.4, a decision with no expiry condition is a claim about a moment that has passed:

> **Revisit when:** a modelling group is contracted to deliver models (FMI); a second bench or second team appears (`C` becomes 1, and at 3 the arithmetic flips); the greenhouse is instrumented with equipment that speaks OPC UA; anybody external asks to review the evidence (VV-40 and Sec. 13.2.2's given case); or a learned model reaches production (Sec. 13.3.6).

13.7 Faded example: the turbine fleet and its auditor

Chapters 4 through 12 each took the turbine one step further. Now decide its standards. Two parts are worked; four are yours.

The system, recapped. Eighty floating offshore wind turbines, a platform-based build (Chapter 12 Sec. 12.9), four of Chapter 3's five services, and two outputs at two levels of rigour – maintenance scheduling, advisory, and life extension, which faces a regulator. Diagnostic twins of this shape run on operational assets [33] and are validated against full-scale prototypes [34].

(a) The counterparty count is not zero, and that settles most of it – worked. Count them: the turbine manufacturer, who supplies models; the certification body, which reviews the life-extension case; the operator's own asset-management system; two maintenance contractors; and, at fleet scale, other turbines' twins. `C` is comfortably above

three for interface standards and **at least one of the counterparties is a regulator**, which puts the certification path into Sec. 13.2.2's *given* category where the arithmetic does not run.

So the decision inverts almost completely from the demonstrator's, and the reason has nothing to do with the technology being harder. It is that somebody else is at the other end of every interface.

(b) FMI is the clear win here – worked, because it shows what r means for a standard. The manufacturer supplies structural models. Without FMI that is a bilateral integration per model per version, negotiated and maintained. With it, the model arrives as something the runner executes. Chapter 6 Sec. 6.8's model-exchange versus co-simulation distinction now becomes a **procurement** question – which variant does the contract require, and therefore who owns the solver – and Chapter 7 Sec. 7.7.5's credibility clauses are what you attach to it, because FMI standardises the interface and says nothing about whether the model deserves belief.

Now yours.

(c) Run Sec. 13.3.7's table for this asset and give each standard a decision with a reason. At least one should be *given* rather than chosen; say which and what it obliges you to produce.

(d) Sec. 13.3.5 noted that VV-40 assumes a one-off credibility assessment, which Chapter 8 Sec. 8.6.1 found unresolved for data-driven models. This twin has a learned anomaly detector (Chapter 8 Sec. 8.8). Say what you would present to the certification body for that component, and whether you would put it in the certification path at all.

(e) Apply Sec. 13.4.5's first item. What export format would you want the fifteen-year evidence trail in, given Chapter 10 Sec. 10.10(b)'s finding that a 2026 retention decision can invalidate a 2041 certification?

(f) Sec. 13.5.3's exemplars are all small systems. Say what an exemplar for a fleet of this kind would have to contain to be useful, and why one probably does not exist. *Hint: what does Chapter 12 Sec. 12.9's arithmetic imply about who can afford to build one?*

13.8 Posed problem: the standards clause

No solution is given.

The situation. The water utility of Chapters 9 to 12 is procuring the next phase of its district-heating twin. Its legal team has sent you a draft clause:

> *"The supplier shall deliver a digital twin compliant with all relevant international standards, including but not limited to ISO 23247, and shall ensure full interoperability with the Authority's existing systems."*

You have been asked to fix it. You also know: the utility exchanges data with two metering vendors and one regional grid operator; an economic regulator reviews its capital programme every five years; and the internal Geographic Information System (GIS) team maintains its own asset register.

Produce a revised clause and a supporting note of no more than four pages containing:

1. **What is wrong with the draft clause**, in three specific points. At least one should concern the difference between a vocabulary standard and an interface standard (Sec. 13.2.1), and at least one the word "compliant" given Sec. 13.3.1's finding about measurability.
2. **A counterparty count** (Sec. 13.1.3), naming each and saying whether they already implement anything.
3. **A decision per standard** in Sec. 13.3.7's table, marked chosen or given, with the arithmetic where it applies.
4. **A revised clause**, which should be shorter than the original, name specific standards for specific exchanges, and be **checkable** – Chapter 2 Sec. 2.11's rule that a term nobody can check is not a term.
5. **A test plan for the interoperability claim**, per Sec. 13.1.2: what exchange you will actually run, with whom, before acceptance.
6. **What you will require of any open-source dependency** the supplier uses, per Sec. 13.5.2's four checks.
7. **The evidence obligation** for the regulator, per Sec. 13.2.2 – and say plainly whether the utility already has it, given Chapter 10 Sec. 10.11's post-mortem.
8. **One paragraph on what you are deliberately not requiring**, and why requiring it would cost the utility money for nothing.

What a good answer looks like. It replaces "all relevant international standards" with a short list tied to named exchanges, because an unbounded obligation is unpriceable and every supplier will price it as risk. It notices that ISO 23247 cannot deliver interoperability and that requiring it *for that purpose* is a category error the supplier will happily satisfy without helping. It makes the interoperability claim testable rather than asserted. It treats the regulator as a given standard and asks what must be produced rather than which standard names it. And its final paragraph is honest that the biggest risk here is not a missing standard but Chapter 10's missing provenance, which no clause about standards will fix.

13.9 Summary

Seven things, tied to the five objectives.

1. **Chapter 1's promise, completed** (Sec. 13.1.1). Re-inventing a defined interface is cost with no value attached *when somebody else is already using that definition.* When nobody is, adopting it is also cost with no value attached, and usually the larger of the two.
2. **The field says the landscape does not yet deliver what standards are for** (Sec. 13.0, Sec. 13.1.2): ISO 23247 adoption is embryonic and its compliance is not measurable; description formats are mutually incompatible; AAS implementations disagree with each other. **So test the exchange, never infer it from the name.**
3. **Four kinds, and a division that matters more** (Sec. 13.2). Vocabulary, interface, process, regulation – and **chosen versus given**. A given standard is not subject to any arithmetic; the only question is what it obliges you to produce, and

the answer is usually evidence you should have anyway in somebody else's shape. *(Objective 1.)*

4. **Six standards, explained rather than named** (Sec. 13.3), each with what it is, what `F` it costs, and the condition under which it pays – including the two Chapter 7 Sec. 7.13 said had been "named once each and did not explain". Their cost ordering is stable even though the numbers are illustrative: vocabulary cheap, interfaces moderate, asset description expensive, process cheap to read and expensive to satisfy. *(Objective 2.)*
5. **The counterparty rule decides it** (Sec. 13.1.3, Sec. 13.4.3): adopt when `C x I > F`. For the demonstrator `C` = 0 and the answer is nothing; for a supplier with five customers the same standard at the same cost is an unambiguous yes. **Same standard, same cost, opposite decisions, and the only variable is the number of people you must agree with.** *(Objective 3.)*
6. **The audit is the uncomfortable part, and the accounting behind it is the interesting part** (Sec. 13.4.2, Sec. 13.6.3). Run all six standards against Chapter 1's four build lines and they remove about one engineer-day for 25 to 50 days of adoption cost. Ordinary open-source components look like they remove six and a half – but they do not, because **`B` = 30 already presupposes them; they are the reason it is 30 rather than hundreds.** The re-use that pays most is invisible in every estimate, because an estimate counts only what somebody considered building; the re-use that is visible is the one that must justify itself. *(Objective 3, 4.)*
7. **The exemplars are the chapter's best answer to its own title** (Sec. 13.5.3). A standard saves you an agreement, a framework saves you code, and an exemplar saves you the weeks you would spend discovering what this book has spent twelve chapters telling you. The incubator, GreenhouseDT and the demonstrator itself are published, documented and readable, and depending on them costs nothing because it is not a dependency. *(Objective 5.)*

And the note this chapter adds to Part III. Chapters 11 and 12 each concluded that the question is never whether a thing would work but whether a decision needs it. Chapter 13 finds the same answer with a different variable: **not whether a standard is good, but how many people you have to agree with.** Every chapter of this Part has now reduced a technology question to a question about the situation – which is the difference between engineering and shopping, and it is the last thing Part III has to teach before Chapter 14 asks what happens once the thing is running.

13.10 Exercises

Solutions or hints follow. Each exercise names the objective it exercises.

13.10.1 *(Objective 1.)* Classify each as vocabulary, interface, process or regulation, and say whether it would be chosen or given: (a) a customer's purchase order requiring ISO 23247 terminology in all deliverables; (b) a decision to package your model as an FMU so a partner can run it; (c) a sector regulator's requirement for a documented validation process; (d) an internal decision to describe assets using AAS submodels.

13.10.2 *(Objective 1.)* Your project lead says "let's be ISO 23247 compliant so our twin can interoperate with the customer's Manufacturing Execution System (MES)". Give the

two-sentence correction, and say what you would propose instead.

13.10.3 *(Objective 2.)* For each, name the standard from Sec. 13.3.7 that would help and state the condition under which it pays: (a) a supplier will deliver three thermal models a year and you must run them; (b) an insurer will audit your remaining-life predictions; (c) you must publish machine-readable descriptions of 200 pieces of equipment to a customer's procurement system; (d) your plant's PLCs already expose a rich data model and you must read it.

13.10.4 *(Objective 3.)* Compute the counterparty decision for a standard with $\texttt{F} = 18$ engineer-days where a bilateral integration costs $\texttt{I} = 5$, for $\texttt{C} = 1$, 3 and 6. Then state which of the three answers you would be least confident in and what you would measure to settle it.

13.10.5 *(Objective 3.)* A colleague argues that adopting AAS now is worth it "because we'll have more partners in three years". Using Sec. 13.4.4, name which counting error this is, and write the alternative that gets them most of what they want.

13.10.6 *(Objective 3.)* Run Sec. 13.4.2's audit shape for a twin whose build is: connector 20 days, model 15, services 25, deployment 10. The physical twin speaks OPC UA natively and two suppliers deliver FMUs. Estimate `r` for each of those two standards against those lines, and say whether the answer would change if the physical twin spoke a proprietary protocol instead.

13.10.7 *(Objective 4.)* For each dependency, say which of Sec. 13.5.1's three kinds it is and which of Sec. 13.5.2's four checks matters most: (a) a widely used time-series database; (b) a twin framework from a university group with two active contributors; (c) a published incubator case study you are reading for ideas; (d) a plotting library that has not had a release in three years.

13.10.8 *(Objective 4.)* You need a change in a twin framework maintained by one research group. Compare forking, carrying a patch and contributing upstream, in engineer-days over three years, stating your assumptions. Then say which of Chapter 12 Sec. 12.6.2's seams reduces the cost of all three.

13.10.9 *(Objective 5.)* For each of the three exemplars in Sec. 13.5.3, name one chapter of this book whose content you could check against it, and one question it would *not* answer.

13.10.10 *(Objectives 1-5, and the one that leaves the book.)* Read one of Sec. 13.5.3's exemplars end to end – the incubator is the most complete. Then write two lists: three things it does that this book recommends, and three things it does differently. For each of the second three, decide whether it is a difference of situation or a disagreement, and say which of Chapter 1's questions would settle it.

Solutions and hints

13.10.1. (a) Vocabulary, **given** – and note it obliges you to produce terminology, not interoperability. (b) Interface, chosen – and the counterparty count is at least one, so run Sec. 13.1.3. (c) Process, given. (d) Interface, chosen – **and the word "internal" should make you check `C`**, because an internal decision with zero counterparties is Sec. 13.4.3's demonstrator case wearing an Industry 4.0 badge.

13.10.2. "ISO 23247 is a vocabulary standard – it defines domains, roles and terms and

specifies nothing on the wire, so conforming to it cannot make two systems interoperate; and 'compliant' is a claim we could not verify anyway, since measuring compliance with it is not currently possible [1]. What will make us interoperate with the MES is agreeing a specific exchange – most likely OPC UA if their system speaks it – and testing it before we commit, so let us use ISO 23247's vocabulary in the documents and put the interoperability requirement where it belongs."

13.10.3. (a) FMI, and it pays because the models cross an organisational boundary three times a year – the counterparty count is one supplier but the integration recurs, which is the case where `I` should be counted per delivery rather than once. (b) ASME VV-40, and it pays because an external reviewer exists; note this is Sec. 13.2.2's given case if the insurer specifies it. (c) AAS, and it pays because `C` is at least one and the exchange is exactly what it is for – but run Sec. 13.3.4's caution about implementation incompatibility first. (d) OPC UA as a client, and it pays immediately because the alternative is a translation layer you own forever.

13.10.4. `C x I` versus `F` = 18. At `C` = 1: 5 < 18, do not adopt. At `C` = 3: 15 < 18, **do not adopt, but only just**. At `C` = 6: 30 > 18, adopt. **The least confident answer is `C` = 3**, because it is within the error of both estimates – and the thing to measure is `I`, by actually doing one bilateral integration and timing it, which you will have to do anyway under the do-not-adopt branch. That is the cheap experiment: the first integration settles the parameter that decides the rest.

13.10.5. The second error in Sec. 13.4.4: **counting your future self**, or in this case future parties. The alternative that gets them most of what they want: **write it as an expiry trigger**, per Chapter 12 Sec. 12.8.4 – "adopt AAS when a second organisation that already implements it needs asset descriptions from us" – and in the meantime keep Chapter 12 Sec. 12.6.2's fourth seam clean, so the context model is data rather than code and can be projected into submodels later. **Cost now: zero. Cost of the option: also close to zero, because Chapter 10 already required that seam.**

13.10.6. Build lines total 70 days. OPC UA against the connector line: the physical twin speaks it natively, so the alternative is writing a protocol translator – `r` might be 8 to 12 of the 20 connector days, against `F` of about 5 as a client. **Clear adopt.** FMI against the model line: two suppliers delivering models, so `r` is the bilateral integrations you avoid rather than model-writing days – say 6 to 10 days over the first year, against `F` of about 1 to consume. **Clear adopt.** If the physical twin spoke a proprietary protocol instead, OPC UA's `r` collapses to zero and the answer flips, **while FMI's is unchanged** – because the two standards are paid for by different counterparties, which is the point of the exercise.

13.10.7. (a) Ordinary component; check 3, the licence, is usually the only live question. (b) Twin framework; **check 2 dominates** – who maintains it and what forking would cost – and Chapter 12's seams are the mitigation. (c) Exemplar; **none of the four checks applies**, because it is not a dependency, and that is the point of Sec. 13.5.1's third row. (d) An ordinary component that is behaving like a framework: three years without a release makes check 2 the live one, and check 1 – does it still work with your stack – the immediate one.

13.10.8. *Hint.* Reasonable assumptions: the change is 3 days to write either way; contributing adds a review cycle, say 2 days spread over weeks; forking means merging

upstream twice a year at 1 to 2 days a time, so 6 to 12 days over three years, plus the risk that a security fix arrives when nobody is free. Carrying a patch is the same merge cost with more surprise. **Contributing is cheapest in every plausible parameterisation**, which is why the section makes the engineering case rather than the ethical one. The seam that reduces all three: **the model interface**, Chapter 12 Sec. 12.6.2's second – if the framework is behind a seam you control, the change may not need to be in the framework at all.

13.10.9. The incubator: check Chapter 7's calibration against [30]'s two-versus-four-parameter comparison; it will not answer anything about scale, since it is one device. GreenhouseDT: check Chapter 10's context history against a system where plants genuinely move; it will not answer Chapter 7's credibility question, which is not its subject. The plant-controller demonstrator: check Chapter 9's connector against a real API with a real history parameter; **it will not answer anything about the digital twin at all**, because the twin is what this book asks you to build – which is the honest and slightly awkward status of the book's own running example.

13.10.10. No solution. One prediction: at least one of your "does differently" items will turn out to be a difference of situation – a different `C`, a different value metric, a different physical twin – rather than a disagreement, and finding that out is worth more than either list.

13.11 Where to go next

In this book. Chapter 14 is the immediate sequel and takes the things this chapter treated as static and puts them in motion: a standard gets a new version, a framework you depend on stops being maintained, an auditor's expectations change, and the credibility argument Chapter 7 wrote expires. Chapter 15 is where Sec. 13.1.3's counterparty count becomes the subject rather than a parameter – ecosystems, twin-of-twins, and exchange across organisational boundaries, which is exactly the regime where the arithmetic of this chapter flips from *skip* to *adopt*. Chapter 15 is also where data spaces land, which Chapters 10 and 12 both deferred.

In the literature, if you want more.

- *The honest notes, and worth reading in full:* [1] analyses 29 manufacturing twin architectures against ISO 23247 using a literature review, a survey and expert interviews, and is candid about how early adoption is; [2] on incompatible twin-description formats; [3] on incompatible open-source AAS implementations.
- *ISO 23247 itself, as used:* [8] is the clearest short analysis and names the Verification, Validation and Uncertainty Quantification (VVUQ) gap; [12] and [13] apply it; [10] builds a component catalogue on it; [4] gives the honest abstraction-level assessment; [35] surveys the proliferation of reference models more generally.
- *Asset description:* [19] is the specification; [18] on mapping between AAS and OPC UA; [36] and [37] on integration work around it; [38] and [39] on applications.
- *The landscape, read for method:* [5] identifies the bodies and their standards, with an explicit Small and Medium-sized Enterprise (SME) framing – **a 2021 deliverable, so use its approach rather than its inventory**. [40] covers the adjacent web-of-things standards world.

- *Interoperability work:* [16] and [17], as in Chapters 9, 10 and 12.
- *Open source and exemplars:* [25] implements the incubator across five frameworks; [29] and [30] are the incubator's own treatments; [31] and [32] are GreenhouseDT; [26], [27] and [28] are working open frameworks.
- *Consulted, not drawn on above:* [41] and [42] on rethinking asset representation, [43] on twin description approaches, [23] on the regulatory framing in healthcare, and [44] on machine-readable trust between twins, which is Chapter 15's direction.

In the demonstrator. Exercise 13.10.10 is the assignment and it is the cheapest one in Part III: read somebody else's twin. The incubator is published, documented and complete, and an afternoon with it will confirm or challenge more of this book than any amount of further reading about standards. **Then do the thing the chapter argues for and count your own counterparties.** If the number is zero, you have just saved yourself several weeks, and that is what this chapter was for.

13.12 References

[1] E. Ferko, A. Bucaioni, P. Pelliccione, and M. Behnam, "Standardisation in Digital Twin Architectures in Manufacturing," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, Mar. 2023, pp. 70–81. doi: 10.1109/ICSA56044.2023.00015.

[2] J. Mattila, R. Ala-Laurinaho, J. Autiosalo, and K. Tammi, "Interoperability of Digital Twins for Automation With Digital Twin Schema," *IEEE Access*, vol. 13, pp. 200595–200608, 2025, doi: 10.1109/ACCESS.2025.3633736.

[3] M. Jacoby *et al.*, "Open-Source Implementations of the Reactive Asset Administration Shell: A Survey," *Sensors*, vol. 23, p. 5229, May 2023, doi: 10.3390/s23115229.

[4] M. Heithoff, N. Jansen, J. Michael, F. Rademacher, and B. Rumpe, "Model-Based Engineering of Multi-Purpose Digital Twins in Manufacturing," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 89–126. doi: 10.1007/978-3-031-67778-6_5.

[5] "Summary of IoT, and DT Standards." Change2Twin project.

[6] "ISO 23247-1:2021," *ISO*. Accessed: Mar. 28, 2025. [Online]. Available: https://www.iso.org/standard/75066.html

[7] T. Böttjer *et al.*, "A review of unit level digital twin applications in the manufacturing industry," *CIRP Journal of Manufacturing Science and Technology*, vol. 45, pp. 162–189, Oct. 2023, doi: 10.1016/j.cirpj.2023.06.011.

[8] G. Shao, S. Frechette, and V. Srinivasan, "An Analysis of the New ISO 23247 Series of Standards on Digital Twin Framework for Manufacturing," American Society of Mechanical Engineers Digital Collection, Sep. 2023. doi: 10.1115/MSEC2023-101127.

[9] N. Anwer, R. Stark, F. Tao, and J. Erkoyuncu, "Developing and leveraging digital twins in engineering design," *CIRP Annals*, vol. 2025, Jun. 2025, doi: 10.1016/j.cirp.2025.05.002.

[10] C. Steinmetz, G. N. Schroeder, R. N. Rodrigues, A. Rettberg, and C. E. Pereira, "Key-Components for Digital Twin Modeling With Granularity: Use Case Car-as-a-Service," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 23–33, Jan. 2022, doi: 10.1109/TETC.2021.3131532.

[11] X. Liu and I. David, "AI simulation by digital twins: Systematic survey, reference framework, and mapping to a standardized architecture," *Software and Systems Modeling*, Aug. 2025, doi: 10.1007/s10270-025-01306-0.

[12] G. Shao, "Use Case Scenarios for Digital Twin Implementation Based on ISO 23247," *NIST*, May 2021, Accessed: Mar. 06, 2024. [Online]. Available: https://www.nist.gov/publications/use-case-scenarios-digital-twin-implementation-based-iso-23247

[13] G. Shao and M. Helu, "Framework for a digital twin in manufacturing: Scope and requirements," *Manufacturing Letters*, vol. 24, pp. 105–107, Apr. 2020, doi: 10.1016/j.mfglet.2020.04.004.

[14] C. Friedrich, A. Lombana, J. Fasquel, C. Schlick, N. Bennani, and M. Mendil, "CoFMPy: A Python Framework for Rapid Prototyping of FMI-based Digital Twins," in *The 2nd International Conference on Engineering Digital Twins*, 2025. Accessed: Nov. 10, 2025. [Online]. Available: https://hal.science/hal-05326255/

[15] G. Schweiger *et al.*, "An empirical survey on co-simulation: Promising standards, challenges and research needs," *Simulation Modelling Practice and Theory*, vol. 95, pp. 148–163, Sep. 2019, doi: 10.1016/j.simpat.2019.05.001.

[16] A. C. Marosi *et al.*, "Interoperable Data Analytics Reference Architectures Empowering Digital-Twin-Aided Manufacturing," *Future Internet*, vol. 14, no. 4, p. 114, Apr. 2022, doi: 10.3390/fi14040114.

[17] M. Picone *et al.*, "Harmonizing Physical and Digital Twins Lifecycles," in *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2025, pp. 197–204. doi: 10.1109/ICSA-C65153.2025.00039.

[18] S. Cavalieri and S. Gambadoro, "Proposal of Mapping Digital Twins Definition Language to Open Platform Communications Unified Architecture," *Sensors*, vol. 23, p. 2349, Feb. 2023, doi: 10.3390/s23042349.

[19] "Asset Administration Shell Part - 1."

[20] M. Viceconti, M. De Vos, S. Mellone, and L. Geris, "Position Paper From the Digital Twins in Healthcare to the Virtual Human Twin: A Moon-Shot Project for Digital Health Research," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 1, pp. 491–501, Jan. 2024, doi: 10.1109/JBHI.2023.3323688.

[21] J. Chae, S. Lee, J. Jang, S. Hong, and K.-J. Park, "A Survey and Perspective on Industrial Cyber-Physical Systems (ICPS): From ICPS to AI-Augmented ICPS," *IEEE Transactions on Industrial Cyber-Physical Systems*, vol. 1, pp. 257–272, 2023, doi: 10.1109/TICPS.2023.3323600.

[22] P. Singh, N. Meratnia, M. Beliatis, and M. Presser, "Navigating the International Data Space To Build Edge-Driven Cross-Domain Dataspace Ecosystem: English," Aug. 2024.

[23] E. Katsoulakis *et al.*, "Digital twins for health: A scoping review," *npj Digital Medicine*, vol. 7, no. 1, pp. 1–11, Mar. 2024, doi: 10.1038/s41746-024-01073-0.

[24] C. Alcaraz and J. Lopez, "Digital Twin: A Comprehensive Survey of Security Threats," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1475–1503, 2022, doi: 10.1109/COMST.2022.3171465.

[25] S. Gil, P. H. Mikkelsen, C. Gomes, and P. G. Larsen, "Survey on open-source digital twin frameworks–A case study approach," *Software: Practice and Experience*, vol. 54, no. 6, pp. 929–960, Jun. 2024, doi: 10.1002/spe.3305.

[26] J. Robles, C. Martín, and M. Díaz, "OpenTwins: An open-source framework for the development of next-gen compositional digital twins," *Computers in Industry*, vol. 152, p. 104007, Aug. 2023, doi: 10.1016/j.compind.2023.104007.

[27] S. Infante *et al.*, "Integrating FMI and ML/AI models on the open-source digital twin framework OpenTwins," *Software Practice and Experience*, Mar. 2024, doi: 10.1002/spe.3322.

[28] P. Talasila, P. H. Mikkelsen, S. Gil, and P. G. Larsen, "Realising Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 225–256. doi: 10.1007/978-3-031-66719-0_11.

[29] C. Gomes *et al.*, "Digital Twin Tutorial: The Incubator Case Study," in *Engineering Trustworthy Software Systems: 6th International School, SETSS 2024, Chongqing, China, April 14–21, 2024, Tutorial Lectures*, J. P. Bowen, C. Gomes, and Z. Liu, Eds., Singapore: Springer Nature, 2025, pp. 68–101. doi: 10.1007/978-981-96-4656-2_3.

[30] B. J. Oakes *et al.*, "Case Studies in Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 257–310. doi: 10.1007/978-3-031-66719-0_12.

[31] E. Kamburjan *et al.*, "GreenhouseDT: An Exemplar for Digital Twins," in *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, in SEAMS '24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 175–181. doi: 10.1145/3643915.3644108.

[32] E. Kamburjan, N. Bencomo, S. L. Tapia Tarifa, and E. B. Johnsen, "Declarative Lifecycle Management in Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, Linz Austria: ACM, Sep. 2024, pp. 353–363. doi: 10.1145/3652620.3688248.

[33] F. Stadtmann and A. Rasheed, "Diagnostic Digital Twin for Anomaly Detection in Floating Offshore Wind Energy." arXiv, Jun. 2024. doi: 10.48550/arXiv.2406.02775.

[34] E. Branlard, J. Jonkman, C. Brown, and J. Zhang, "A digital twin solution for floating offshore wind turbines validated using a full-scale prototype," *Wind Energy Science*, vol. 9, no. 1, pp. 1–24, Jan. 2024, doi: 10.5194/wes-9-1-2024.

[35] J. Pfeiffer *et al.*, "Towards a Unifying Reference Model for Digital Twins of Cyber-Physical Systems." arXiv, Jul. 2025. doi: 10.48550/arXiv.2507.04871.

[36] C. Schmidt, F. Volz, L. Stojanovic, and G. Sutschet, "Increasing Interoperability between Digital Twin Standards and Specifications: Transformation of DTDL to AAS," *Sensors*, vol. 23, p. 7742, Sep. 2023, doi: 10.3390/s23187742.

[37] C. Schmidt, F. Volz, L. Stojanovic, and H. Kett, "Integration Approaches for Digital Twins in Dataspaces," *Applied Sciences*, vol. 15, no. 21, p. 11623, Jan. 2025, doi: 10.3390/app152111623.

[38] F. Schnicke, T. Kuhn, and P. O. Antonino, "Enabling Industry 4.0 Service-Oriented Architecture Through Digital Twins," in *Software Architecture*, H. Muccini, P. Avgeriou, B. Buhnova, J. Camara, M. Caporuscio, M. Franzago, A. Koziolek, P. Scandurra, C. Trubiani, D. Weyns, and U. Zdun, Eds., Cham: Springer International Publishing, 2020, pp. 490–503. doi: 10.1007/978-3-030-59155-7_35.

[39] Q.-D. Nguyen, Y. Huang, F. Keith, C. Leroy, M.-T. Thi, and S. Dhouib, "Manufacturing 4.0: Checking the Feasibility of a Work Cell Using Asset Administration Shell and Physics-Based Three-Dimensional Digital Twins," *Machines*, vol. 12, p. 95, Jan. 2024, doi: 10.3390/machines12020095.

[40] L. Sciullo, L. Gigli, F. Montori, A. Trotta, and M. Felice, "A Survey on the Web of Things," *IEEE Access*, vol. 10, pp. 1–1, Jan. 2022, doi: 10.1109/ACCESS.2022.3171575.

[41] C. Ellwein *et al.*, "Rethinking Asset Administration Shell Communication Types: A Systematic Mapping Study and Portfolio-Based Classification," *Production Engineering*, vol. 20, Dec. 2025, doi: 10.1007/s11740-025-01378-3.

[42] K. Gleich, S. Behrendt, M. Hörger, M. Benfer, and G. Lanza, "An Asset Administration Shell-Based Digital Product Passport as a Gaia-X Service," *Procedia CIRP*, vol. 127, pp. 224–229, Jan. 2024, doi: 10.1016/j.procir.2024.07.039.

[43] J. Zhang, C. Ellwein, M. Heithoff, J. Michael, and A. Wortmann, "Digital twin and the asset administration shell," 2024, Accessed: Apr. 09, 2025. [Online]. Available: https://awortmann.github.io/downloads/paper/Digital_twin_and_the_asset_administration_shell.pdf

[44] "Assuring Trustworthiness in Dynamic Systems Using Digital Twins and Trust Vectors." Digital Twin Consortium, May 2024. Available: https://www.digitaltwinconsortium.org/assuring-trustworthiness-in-dynamic-systems-using-digital-twins-and-trust-vectors-form/