Examining the Role of Context in Data Interoperability

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Jim Smith, Patrick Place, Marc Novakouski, David Carney
October 26, 2011

Overview

Background

Research Approach

Discussion

Conclusions

Next Steps

Background

Motivation for research

- Interoperability continues to be a problem for the DoD
  - Emphasis has been on “filling the gaps,” or imposing commonality (i.e., lexicons, data models, etc.)
- The evolution towards looser coupling between systems of systems constituents (i.e., moving from “directed” to “acknowledged” and “collaborative” systems of systems) has highlighted the limitations with current approaches

The challenges are many

- Most data-centric systems cannot be easily made to interoperate: they weren't designed to work together, or can do so only over a limited range
- This lack of interoperability causes the warfighter to employ workarounds (e.g., “fat fingering” data between incompatible systems)
- The systems of systems that result are brittle – any change (to one or more of the constituent systems, or their operational employment) may “break” interoperability

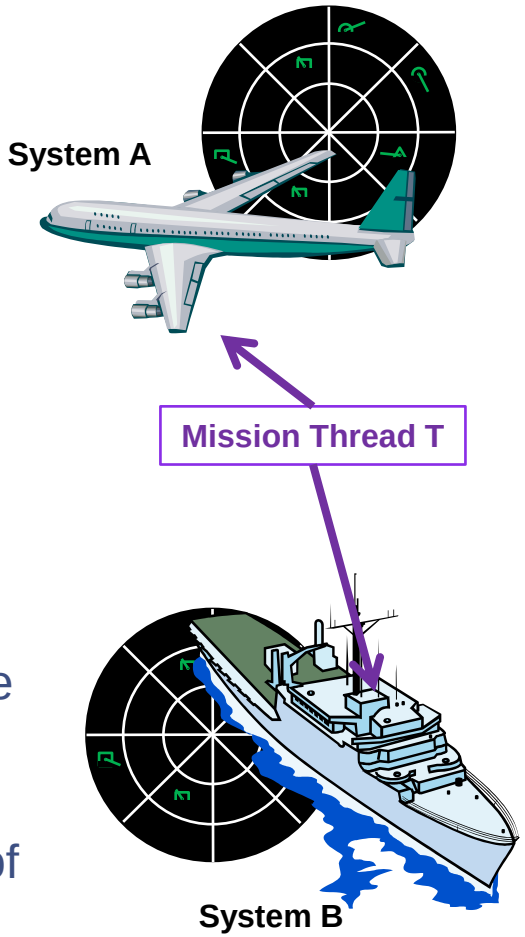
What is needed is a way to reduce time and cost to achieve data interoperability in a system of systems context, while allowing for greater independence of the constituent systems

- *Without* requiring common data models, ontologies, lexicons, etc.

Background: Guiding Scenario - 1

Take a really simple example (drawn from real life):

- Two systems (A and B) process and displayed track information
  - System A is on an airplane; system B is on a ship
  - Both systems are “interoperable,” and have all the usual and customary documentation
- A mission thread (T) requires A and B to exchange air track information to create a common tactical picture
- Using a standard messaging format over a common tactical data link, they exchange track data, and integrate those tracks held by the other system into a common view
  - Interoperability is achieved in a specific instantiation of mission thread T



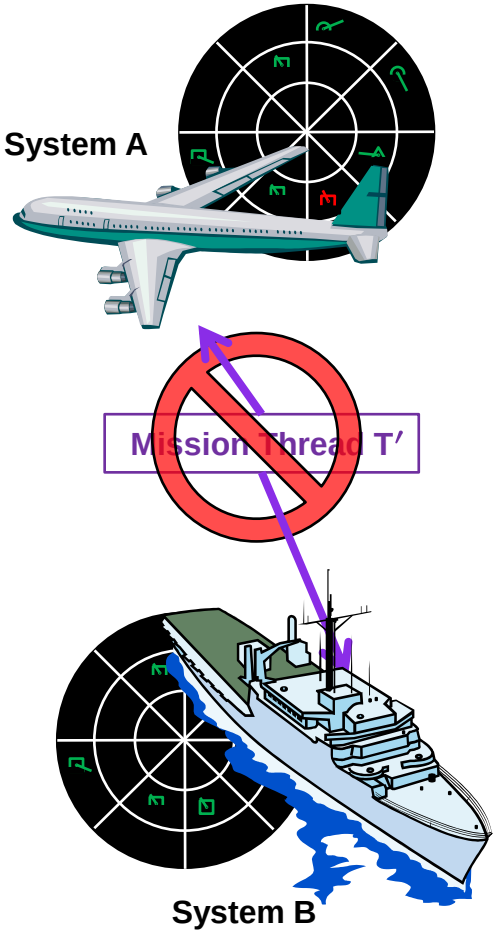
Background: Guiding Scenario - 2

What happens when we change the mission thread context to include surface tracks (designated by T′)?

- Though not apparent from the available documentations, system A didn't correctly handle surface tracks
  - There was an unstated contextual constraint on system A that all tracks were assumed to be air tracks. As a consequence, system A's data models, ontology, interfaces, etc. didn't account other types of tracks
  - Ironically, for compatibility with the messaging format, system A **did** force the appropriate message field to "air track" when passing track information (resulting in system B seeing "phantom" air tracks duplicating surface tracks)
    - Discovering this required reading detailed system specifications, technical requirements, and (ultimately) source code for system A

Had the mission thread context remained air tracks only, then systems A and B would have remained interoperable *in the original context* (T)

This unrecognized contextual limitation resulted in a failure of interoperability – no common tactical picture



Research Approach

Question: Can we represent context in a mathematically rigorous manner so that it can be used to improve understanding and reasoning about interoperability?

Approach:

- Characterize the relationship between context and interoperability
- Identify the aspects of context that are necessary to understand interoperability
  - Develop a schema to represent context
- Develop a mathematical framework for expressing and reasoning about context and interoperability
- Test this approach with a *very* simple case
  - Does it work???

Context - 1

As illustrated by the guiding scenario, a seemingly insignificant change in the system of systems (SoS) operational environment, patterns of use, or in the constituent systems can have a major impact on interoperability

- In SoSs, *all* of these are continually changing

Conventional representations of interoperability (e.g., DoDAF views OV-6, SV-6) focus on the relationships between the constituent systems: filling the gaps in data formats, information exchange requirements, etc.

- The SoS context emerges as a by-product of these views
- As a result, the consequences of a change in the SoS context may not be immediately apparent

Similarly, conventional representations of context focus on the "situation of an entity," and don't encompass many of the factors that are important to understanding interoperability, including:

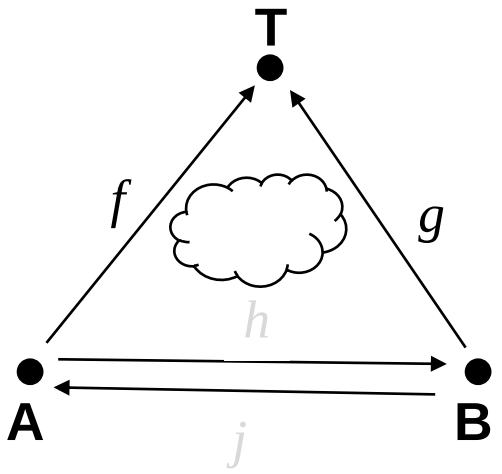
- Constraints on permissible behaviors and operational environment
- Pragmatic aspects of interoperable data exchanges
- Assumptions made by the developers, users, etc.

Context - 2

What if we make context a “first class” object, as opposed to something that is inferred?

Doing so allows us to shift the focus from “filling the gaps” between systems to understanding how systems relate to the SoS operational thread context

- The relationships between the systems - “the gaps” - become a byproduct of the relationships between the systems and the SoS context, rather than the other way around



Context - 3

The context, T, has an ontology, data models, functions/behaviors, and constraints - just like any system

- For convenience, we can represent these with a “contextual schema,” consisting of a “signature” and constraints:
  - This signature is analogous to the signature of an algebra or a logic system, and consists of
    - O : Ontology for the domain of discourse
    - D: Data models for the information exchanges
    - F: Functional behaviors over the ontology and data models
  - The constraints define any restrictions on the allowable behaviors for the
    - C: Contextual constraints
- Combining the signature and constraints, we obtain

$$Sch_x = (Sig_x, C_x) = (O_x, D_x, F_x, C_x)$$

Context - 4

We can also see that the SoS context is separate and distinct from the context of any of the constituent systems:

System of Systems	System
O_T: Ontology for *SoS* domain of discourse	O_{Sys}: Ontology for *system* domain of discourse
D_T: Data models for the *SoS information exchanges*	D_{Sys}: *System* data models
F_T: Set of functional behaviors that implement an SoS *operational thread*	F_{Sys}: *System* functional behaviors
C_T: *SoS operational thread* contextual constraints	C_{Sys}: *System* contextual constraints

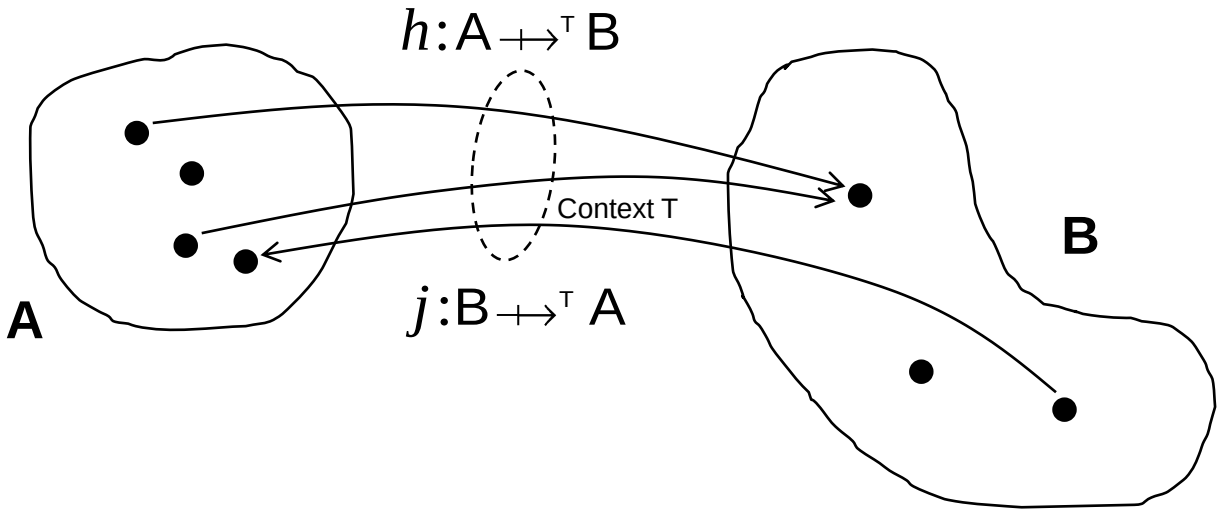
Note: $C_T \neq \sum (C_{Sys_i})$ (likewise for O,D, and F)

$\therefore Sch_T \neq \sum (Sch_{Sys_i})$

Interoperability - 1

Interoperability can be expressed as mappings between *logical systems*:

- Interoperability represents truth-preserving mappings from the contextual schema of one logical system to another or, as previously shown, between the systems and the SoS



Note: these mappings are *partial* functions—interoperability doesn't require that everything be mapped from one logical system to another, *just those aspects that are relevant to a specified context*

Category Theory “Lite” - 1

By expressing interoperability in this way, we are able to leverage a branch of mathematics, known as *category theory*, to simplify how we define and reason about interoperability

Category theory provides a rigorous approach to understanding the relationships between objects and their morphisms (transformations)

- Objects can be sets, algebras, or even contextual schemas
- Morphisms are mappings between objects that preserve their structure and meet certain requirements (i.e., identity, associativity)

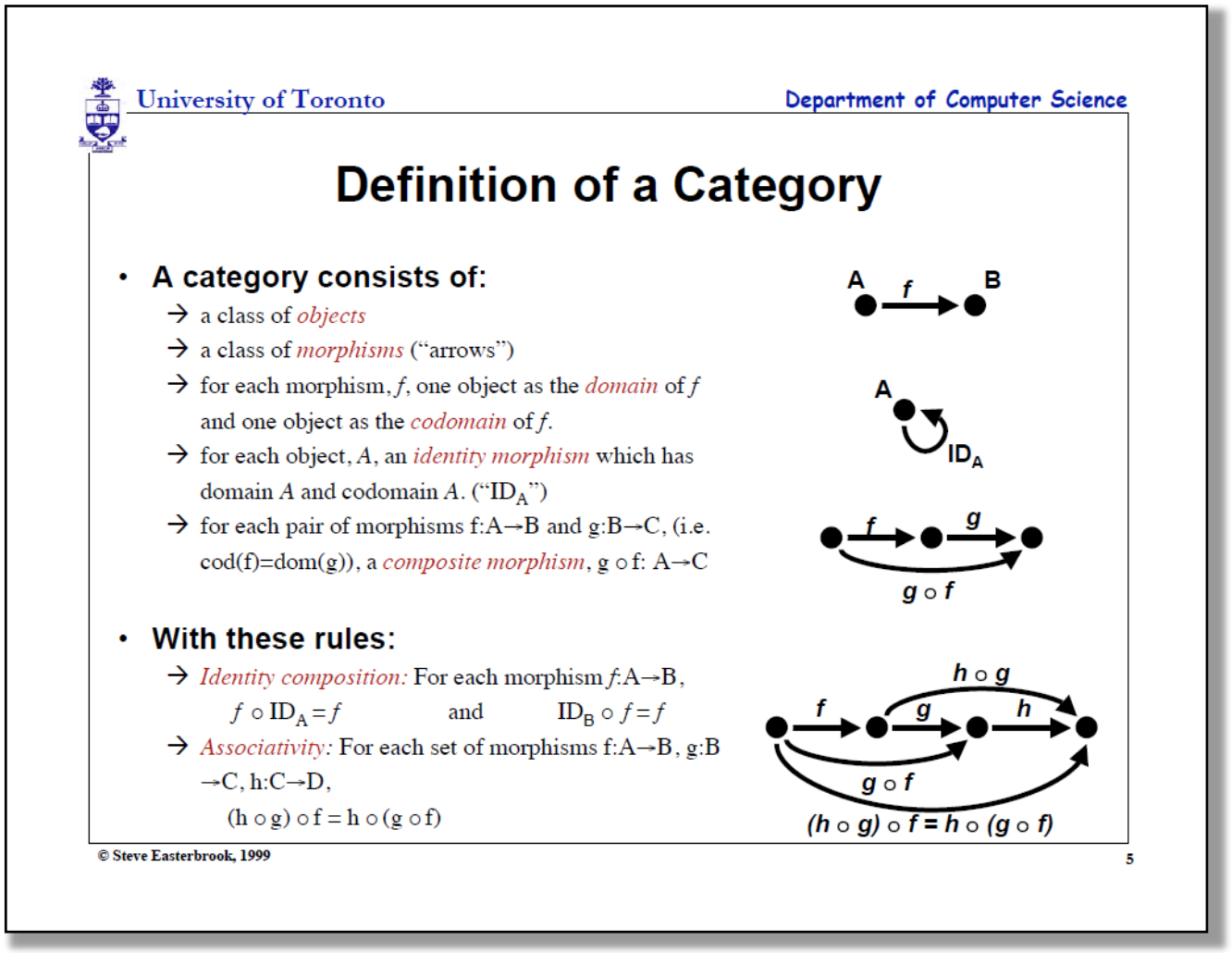
Several uses in software engineering

- Specification transformation and module composition
- Database schema transformation

Provides a powerful tool to “prove” interoperability in a given—or in a new or novel—context

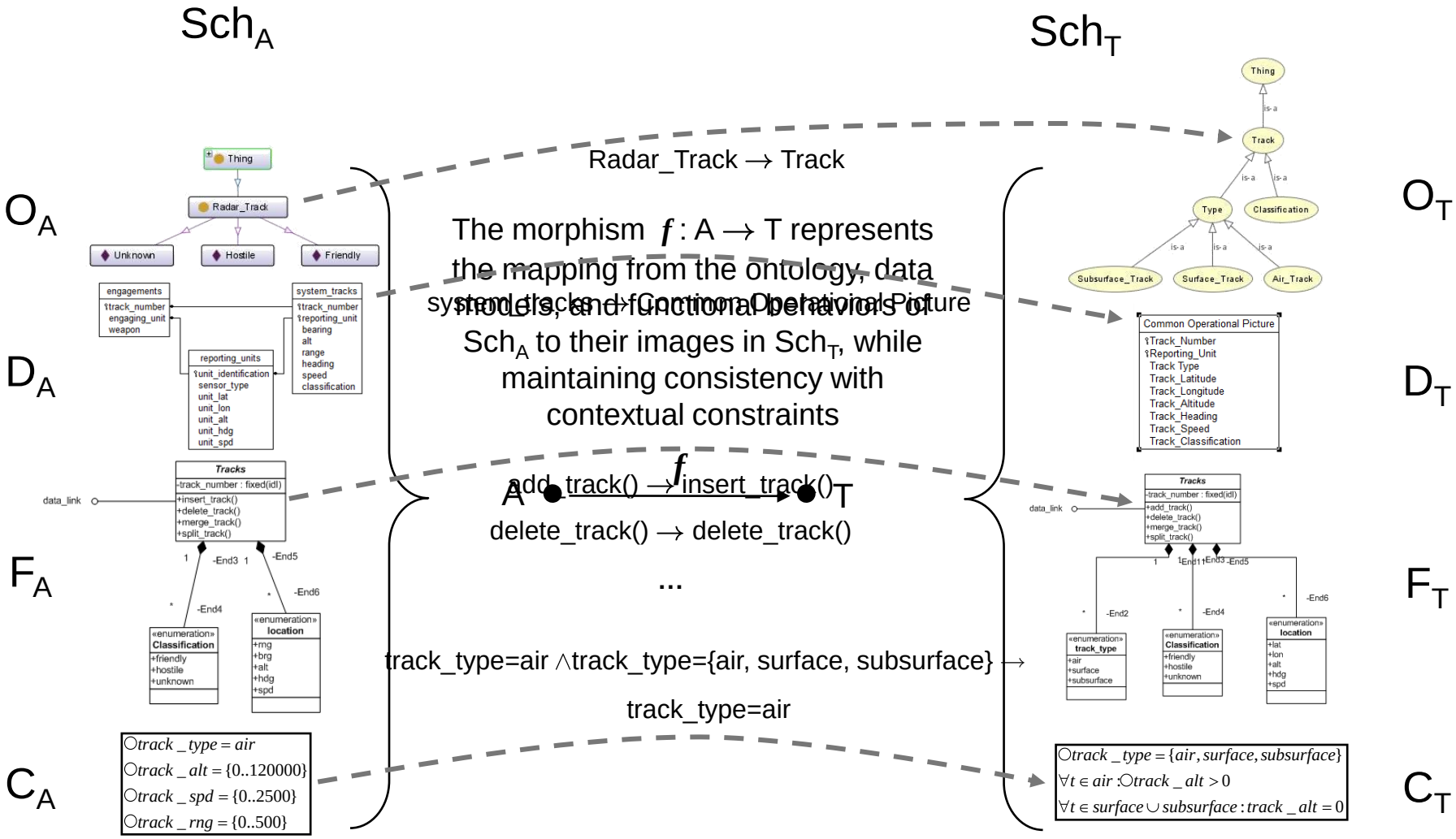
- Allows us to understand what is happening in interoperable information exchanges
  - Both semantic and pragmatic aspects of interoperability
- Helps us identify how and why interoperability has failed

Category Theory "Lite" - 2



From "Category Theory for Software Engineers," Steve Easterbrook, University of Toronto, 1999.

Interoperability - 2



Interoperability - 3

A concrete example can help illustrate this…

We have objects (systems A and B; SoS operational thread T) in the category of contextual schemas:

- A= $\{a_1, a_2\}$ (set of air tracks held by A)
- B= $\{b_1, b_2, b_3\}$ (set of air (b_1, b_2) and surface (b_3) tracks held by B)
- T= $\{t_1, t_2\}$ (set of tracks resulting from the mapping of A and B to the context of T)

and morphisms (mappings between them):

- h : A → B = $\{a_1 \rightarrow b_1\}$ (track reported by A to B)
- j : B → A = $\{b_2 \rightarrow a_2\}$ (track reported by B to A)
- f : A → T = $\{a_1 \rightarrow t_1, a_2 \rightarrow t_2\}$ (The mapping from A to the context T)
- g : B → T = $\{b_1 \rightarrow t_1, b_2 \rightarrow t_2\}$ (The mapping from B to the context T)

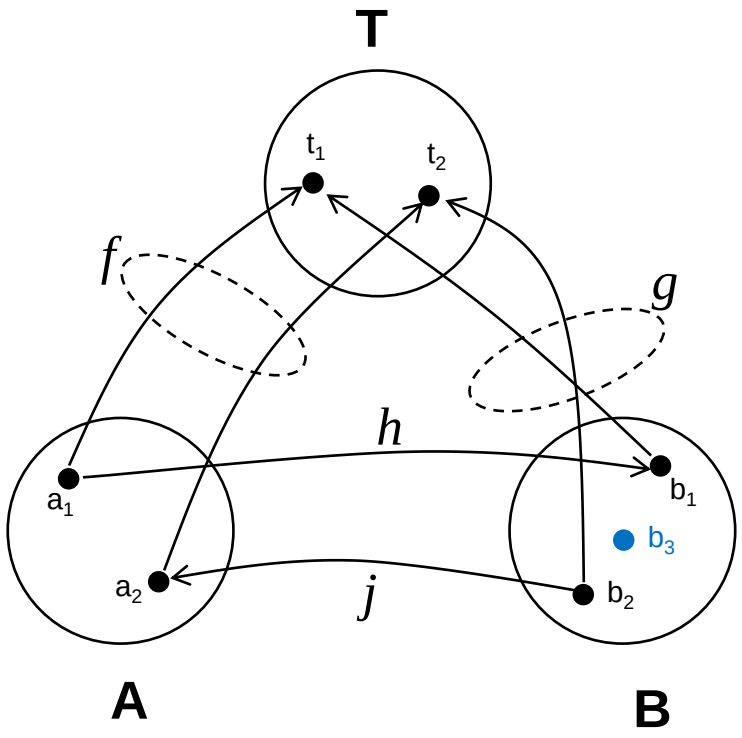
Note: Track b_3 is not reported by B to A because it is not part of context T

By inspection, we see that

A → T = A → B → T (or $f = g \circ h$)

and

B → T = B → A → T (or $g = f \circ j$)



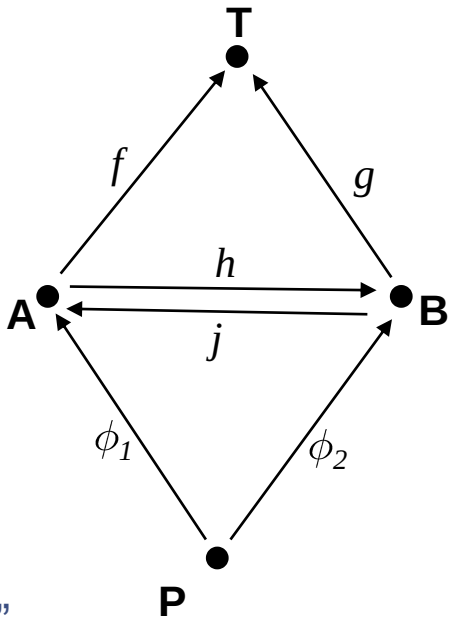
Category Theory "Lite" - 3

From our guiding scenario, A and B represent two systems, and T represents the SoS operational context within which they must interoperate

- Node P (the "pullback* of A and B over T") represents the interoperability between A and B in context T
  - A and B exchange track data in context T so that both systems hold the same tracks

When we changed the SoS context to include surface tracks, interoperability failed

- Since A didn't properly handle surface tracks, A and B had different tactical pictures
  - In this case, P would highlight the fact that the "common" picture was actually not common



* The pullback is a special case of the more general "limit"

Interoperability - 4

Continuing our example…

Construct the pullback, P

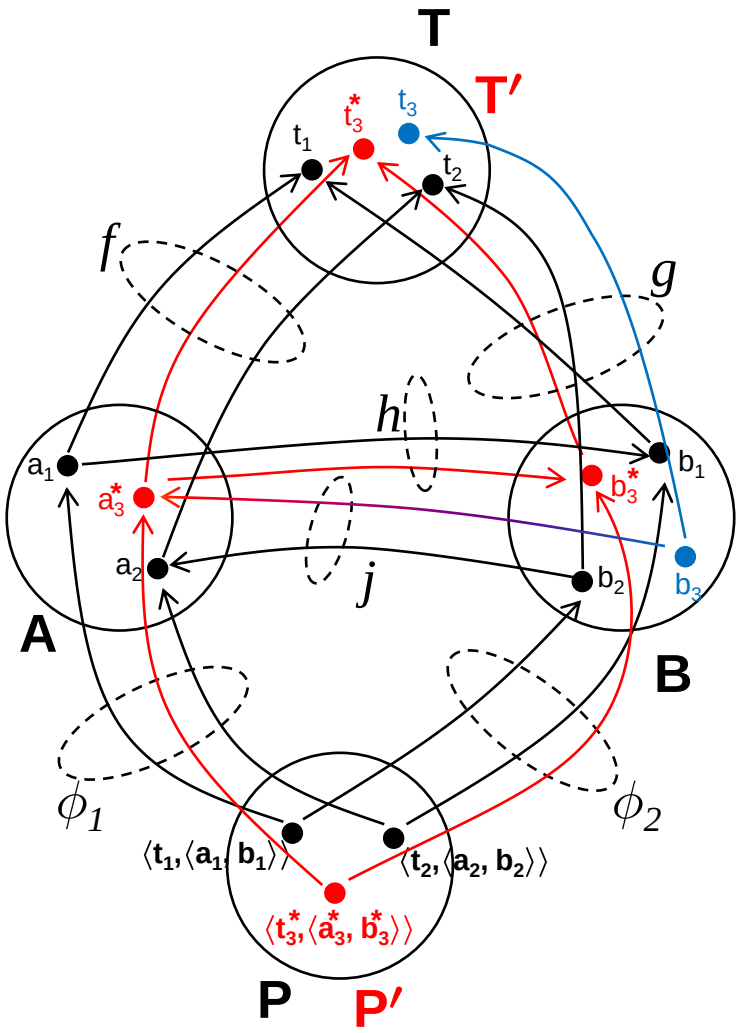
- In context T, P = $\{\langle t_1,\langle a_1, b_1\rangle\rangle, \langle t_2,\langle a_2, b_2\rangle\rangle\}$
- The pullback of A and B over T shows the tracks jointly held by A and B in context T: the common tactical picture

Now, change the mission thread context to include surface tracks (T′), and construct a new pullback (P′):

- b_3 is a surface track held by B which, unlike the previous case, is *now* part of the context (T′)
- A incorrectly interprets it as an air track, a_3^*
- A subsequently passes this invalid track back to B as a "phantom" air track, b_3^*

Thus, in context T′, the revised pullback P′= $\{\langle t_1,\langle a_1, b_1\rangle\rangle, \langle t_3^*,\langle a_3^*, b_3^*\rangle\rangle, \langle t_2,\langle a_2, b_2\rangle\rangle\}$

- Note that new pullback reflects the "phantom" track (t_3^*) resulting from the incorrect interpretation of surface track b_3 by system A – no longer have a common tactical picture



Reasoning Framework - 1

We can represent the contextual schemas using an ontology language (e.g., RDF, OWL), XML schema and applicable DoDAF architecture views:

- OV-3
- OV-6
- SV-6

} various aspects of the data exchange semantics

- SV-10: system constraints/permissible behaviors
- SvcV-6 and SvcV-10: same as SV-6 and SV-10 for services

The Rei ontology* provides a language for expressing constraints as policies, including conflict resolution (e.g., “A should do x, unless it is overridden by a higher-priority demand”)

<deontic:Permission rdf:ID="Perm_StudentPrinting"> <deontic:actor rdf:resource="#PersonVar"/> <deontic:action rdf:resource="#ObjVar"/> <deontic:constraint rdf:resource="#IsStudentAndBWPrinter"/> </deontic:Permission>

Deontic logic then provides a framework to reason about constraints

- Allows us to “prove” that interoperability between systems can exist in a given context, and identify its bounds

* http://www.cs.umbc.edu/~lkagal1/rei/

Reasoning Framework - 2

From our guiding scenario, we define the following database schemas:

System A

```
schema Tracks {
        interface Track {
                int tracknum();
                int lat();
                int lon();
                int alt();
                int hdg();
                int spd();
                Set<Track> tracks();
                }
        Collection<Track> dbTracks;
        Constraints:
        Tracks X,Y;
        tracknum(X) != tracknum(Y);
}
```

System B

```
schema Tracks {
        interface Track {
                int track_number();
                int track_type();
                int track_position();
                int alt();
                int hdg();
                int speed();
                Set<Track> tracks();
                }
        Collection<Track> dbTracks;
        Constraints:
        Tracks X,Y;
        tracknum(X) != tracknum(Y);
        track_type(X) = {surface, air, subsurface}
}
```

Context T

```
schema Tracks {
        interface Track {
                int track_no();
                int track_type();
                int track_lat();
                int track_lon();
                int track_alt();
                int track_head();
                int track_spd();
                Set<Track> tracks();
                }
        Collection<Track> dbTracks;
        Constraints:
        Tracks X,Y;
        tracknum(X) != tracknum(Y);
        track_type(X) = air
}
```

Reasoning Framework - 3

In the original context, the pullback of A and B over T is represented by

Tracks =
<track_no, <tracknum, track_number>>
<air, <air, track_type>>
<track_lat, <lat, position.lat>>
<track_lon, <lon, position.lon>>
<track_head, <hdg, head>>
<track_spd, <spd, speed>>

The track type constraints are given by

(i) $\bigcirc_A air$ (system A must handle "air" tracks)

(ii) $\bigcirc_B \ air \wedge surface \wedge subsurface$ (system B must handle tracks of all three types)

(iii) $\Box_T air$ (it is necessary that tracks be of type "air")

This gives us:

(iv) $\Box_T air \Rightarrow \mathrm{F}_A \neg air \wedge \mathrm{F}_B \neg air$

By inspection, we can see that the constraints on A and B are consistent in this context:

(v) $\bigcirc_A air \Rightarrow \mathrm{F}_A \neg air$

(vi) $\bigcirc_B air \Rightarrow \mathrm{F}_B \neg air$

Reasoning Framework - 4

When we changed the mission thread context to include other track types (i.e., surface and subsurface), we have

```
Context T'
schema Tracks {
        interface Track {
                int track_no();
                int track_type();
                int track_lat();
                int track_lon();
                int track_alt();
                int track_head();
                int track_spd();
                Set<Track> tracks();
                }
        Collection<Track> dbTracks;
        Constraints:
        Tracks X,Y;
        tracknum(X) != tracknum(Y);
        track_type(X) = {surface, air, subsurface}

}
```

The pullback of A and B over T' is represented by

```
Tracks =
        <track_no, <tracknum, track_number>>
        <track_type, <air, track_type>>
        <track_lat, <lat, position.lat>>
        <track_lon, <lon, position.lon>>
        <track_head, <hdg, head>>
        <track_spd, <spd, speed>>
```

Reasoning Framework - 5

This gives us a interoperability limitation

(i) $\bigcirc_A air$ (system A tracks must handle "air" tracks)

(ii) $\bigcirc_B \ air \wedge surface \wedge subsurface$ (system B must handle tracks of all three types)

(iii) $\Box_{T'}(air \wedge surface)$ (it is necessary that tracks include "air" and "surface")

As before, we restate the universal necessity as:

(iv) $\Box_{T'}(air \wedge surface) \Rightarrow \bigcirc_A(air \wedge surface) \wedge \bigcirc_B(air \wedge surface)$

However, since system A only supports air tracks, this violates one of its contextual constraints

(v) $\bigcirc_A air \boxplus \exists \mathrm{F}_A \neg air \Rightarrow \mathrm{F}_A \ surface$

(vi) $\bigcirc_B(air \wedge surface \wedge subsurface) \Rightarrow \mathrm{P}_B(air \vee surface)$

So, even though neither system was changed, a change in mission thread context resulted in loss of interoperability

Conclusions

Treating context as a “first class” object (like a system) allows us to re-think interoperability

- Focus on the relationships between the constituent systems and the SoS context, rather than “filling the gaps” between the systems
- Making context explicit makes it much easier to see how a change in context—either for the SoS, or a constituent system—can affect interoperability
  - SoS and system context can be represented by their contextual schemas

Category theory and deontic logic can be used to reason about interoperability in an SoS context

Next Steps

We've proven it *can* work—for a very simple problem

- Still need to see if this approach can work for more realistic problems
  - Does it scale up?
  - Is the cost greater than the benefits?
  - Is there some way to do this without requiring program managers to have a Ph.D. in applied mathematics???
- Looking for potential candidates to pilot this approach

A category-theoretic approach can also be employed in reasoning about interoperability in a new SoS context, or with a novel composition of systems

- Instead of a pullback/limit, use a pushout/co-limit

Contact Information

Jim Smith

Senior Member of the Technical Staff

Telephone: +1 703-908-8221

Email: jds@sei.cmu.edu

Web: www.sei.cmu.edu/staff/jds

U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Web

www.sei.cmu.edu

www.sei.cmu.edu/contact.cfm

Customer Relations

Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.