Structured Co-spans: An Algebra of Interaction Protocols[†]

José Luiz Fiadeiro and Vincent Schmitt

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{jose,vs27}@mcs.le.ac.uk

Abstract. We extend the theory of (co-)spans as a means of providing an algebraic approach to complex interactions as they arise in software-intensive systems. In order to make interconnections independent of the nature of components involved, interaction protocols are formalised not in terms of morphisms (i.e. part-of relationships) but a generalised notion of (co-)span in which the arms are structured morphisms – the head (the glue of the protocol) and the hands (the interfaces of the protocol) belong to different categories, the category of glues being coordinated over that of the interfaces. The proposed generalisation sheds some additional light into adjunctions in bicategories, namely on the factorisation of left adjoint 2-sided enrichments.

1 Introduction

Software is becoming an integral part of a range of products and services performing vital functions in all sectors of economic and social activity. In such *software-intensive systems*, software applications are required to *interact*, in a seamless way, with other software components, devices, sensors, even humans. The complexity involved in building the software components that will be deployed in such systems in not so much on the "size" of their code but on the number and intricacy on the interactions in which they will be involved, what in [6] we have called *social complexity*. From an algebraic point of view, social complexity raises new challenges with respect to the more established *physiological complexity*, i.e. the fact that a complex whole can be understood as a composition of its parts. The basic difference is that it does not make sense to see software-intensive systems as being compositions, in an algebraic sense, of simpler components. There is not a notion of whole to which the parts contribute but, rather, a number of autonomous entities that interact with each other through external connectors.

This is why it is so important to put the notion of interaction at the centre of research in software-intensive system modelling, and to support methods and languages that separate interaction concerns from computational ones. In the past, we developed

[†] This work was partially supported through the IST-2005-16004 Integrated Project *SENSORIA: Software Engineering for Service-Oriented Overlay Computers*.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 194-208, 2007.

a categorical framework supporting the separation between "computation" and "coordination" as architectural dimensions in software development [9]. This framework is based on what we have called "coordinated categories" [5] – concrete categories (faithful functors) that externalise the interfaces used by components to interact with other components. From an algebraic point of view, we propose to work with "structured morphisms"[1], i.e. pairs $\langle f,S\rangle$ where $f:A\rightarrow \boldsymbol{G}S$ is a $\boldsymbol{C}$-morphism, $A:\boldsymbol{C}$, $S:\boldsymbol{D}$, and $\boldsymbol{G}:\boldsymbol{D}\rightarrow\boldsymbol{C}$. The motivation is that $\boldsymbol{G}$ "forgets" the computational part of the objects of $\boldsymbol{D}$ and returns their interfaces; structured morphisms capture interactions that do not depend on the computational processes involved in components.

Ultimately, the (autonomic) entities that we wish to interconnect need not be organised in a category. Typically, in a category of systems, morphisms capture a "component-of" or "sub-system" relationship. As already motivated, in software-intensive systems it does not make sense to talk about "component-of" relationships in an algebraic way. Therefore, we decided to look for algebraic mechanisms of interconnection that can capture peer-to-peer interactions among autonomous components. That is why, in this paper, we report on the use of co-spans – pairs $\langle f_A,f_B\rangle$ where $f_A:A\rightarrow S$ and $f_B:B\rightarrow S$ are morphisms of a category $\boldsymbol{D}$. Co-spans (and their dual – spans) have been deserving increasing attention in computer science, namely when $\boldsymbol{D}$ is a category of graphs (or variants of graphs) as models of concurrent processes or reactive systems [14] – generalised automata or transition systems in one sense or another. For instance, spans can be used for defining composition operations along interfaces, which capture the behaviour of communicating parallel processes [11].

Because we want the application of interaction protocols to be "agnostic" to the nature of the computations that are performed by the peers, we want that the protocol be based on the interfaces that components have available for interacting with each other, not on the computations that they perform locally. This suggests that the interactions should be established between objects of a category of interfaces, not between behaviours. That is, we should work with co-spans based on structured morphisms – triples $\langle f_A,S,f_B\rangle$ where S is an object of $\boldsymbol{D}$ and $f_A:A\rightarrow \boldsymbol{G}S$, $f_B:B\rightarrow \boldsymbol{G}S$ are structured morphisms of a coordinated category $\boldsymbol{G}:\boldsymbol{D}\rightarrow\boldsymbol{C}$.

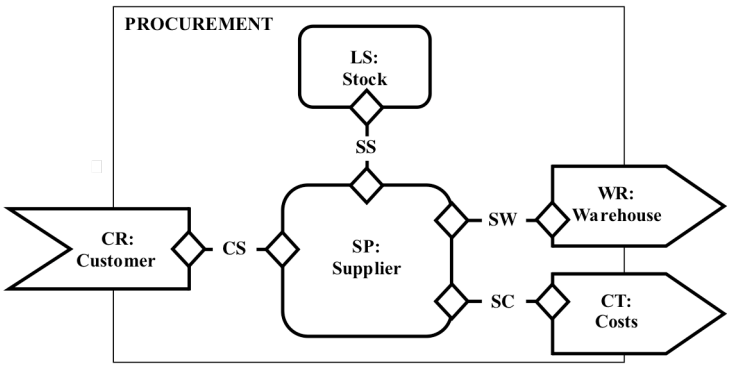
Our approach is also different to the traditional uses of (co-)spans in that we are interested in a more declarative setting in which the objects of $\boldsymbol{D}$ are not operational models of behaviour (automata, transition systems, and so on), but specifications or designs of protocols. This is why we are interested in other categories than that of graphs. In fact, we will work over coordinated categories in general, which include graphs and other models of concurrency, but also logical and algebraic specifications [5,10,13].

Our purpose in this paper is to generalise the theory of co-spans to support an algebraic approach to interactions in software-intensive systems as discussed above. In Section 2, for further motivation, we present a case study that we have been developing for service-oriented modelling in the context of the SENSORIA project. In Section 3, we discuss in more detail the notion of interaction protocol that we have in mind and the role played by structured co-spans. Finally, in Section 4, we investigate the properties of bicategories of structured co-spans, which leads to some interesting new results in adjuncions of 2-sided enrichments.

2 Modules for Software-Intensive Systems

The work that we present in this paper has been inspired by research that we have been developing within the IST-FET Integrated Project SENSORIA – *Software Engineering for Service-Oriented Overlay Computers* – on the emerging service-oriented computing paradigm, generalising methods and techniques already proposed for web-service and grid technologies. From the point of view of software-intensive systems, services can be understood as autonomous, platform-independent computational entities that can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems. In this paper, we do not address the publication and run-time discovery process that characterises the service-oriented paradigm. This is because we concentrate on the static structure of systems, not on the process through which they can be dynamically configured.

The modelling language that we have been defining in SENSORIA – SRML – offers a notion of module through which composite services can be specified as assemblies of internal components and externally procured services [7]. In order illustrate and motivate the notion of module, we use a typical procurement business process involving a supplier *SP*, a warehouse *WR*, a local stock *LS*, a price look-up facility *CT*, and a customer *CR*.



This module declares *SP* and *LS* as components. Components are the computational units that constitute the core of the module and are typed by what we call *business roles*; in the example, *SP* plays the business role of *Supplier* and *LS* of *Stock*. A business role specifies the activity performed by a component in terms of a collection of transitions. As an example (see [7] for an explanation of the syntax), the business role *Stock* models a behaviour pattern that is typical of a database view:

```
BUSINESS ROLE Stock is
   INTERACTIONS
      rpl get(product):nat
      prf set(product,nat)
   ORCHESTRATION
      local qoh:product→nat
      transition
         triggeredBy get(p)
         sends qoh(p)
```

```
transition
   triggeredBy set(p,n)
   effects qoh(p)'=n
```

The model provided through a business role is independent of the language in which the component is programmed and the platform in which it is deployed. The "orchestration", i.e. the specification of the pattern of behaviour exhibited by the component, is independent of the specific parties that are actually interconnected with it in any given run-time configuration; a component is totally independent in the sense that it does not invoke services of any specific co-party – it just offers an interface of two-way interactions in which it can participate. Interconnections with other entities are established through what we call *wires*, as discussed below.

Modules can identify external parties that play a role in the business process – *WR*, *CT* and *CS*. Making certain parties external reflects looser coupling and late binding. For instance, making the warehouse *WR* an external party reflects the fact that the choice of warehouse should probably be made at run-time, e.g. taking into account properties of the customer like its location. Every external party is typed by what we call a *business protocol*, which specifies a stateful interaction between a component and the corresponding party. In SRML, this specification is given in a temporal logic of interactions. As an example (once again, please see [7] for an explanation of the syntax used in business protocols), consider the behaviour required of a warehouse:

```
BUSINESS PROTOCOL Warehouse is
INTERACTIONS
   r&s check&lock
   snd confirm
BEHAVIOUR
   initially check&lock🔔?
   check&lock⌛ ⊃ (check&lock✓? ensures confirm🔔!)
   check&lock✓? ⊃ (check&lock✞? exceptif confirm🔔!)
```

Basically, we are stating that (1) in the initial state the warehouse is ready to receive a request for engaging in the interaction *check&lock* (which the wire *SW* connects to *BA* – the booking agent), (2) the warehouse promises to issue *confirm* if a commitment to the deal proposed by *check&lock* is received within an agreed delay, and (3) the commitment can be revoked until the *confirm* is actually issued. The difference with respect to business roles is that, instead of an orchestration, a business protocol declares the set of properties that the co-party is required to adhere to. Otherwise, both business roles and protocols share the same kind of declaration of the interactions in which they can be involved, what we call their *signatures*.

Modules can offer an external interface for other modules to use its services – *CR* in the case at hand. The corresponding business role specifies constraints on the interactions that the module supports as a service provider such as the order in which they expect invocations to be made or deadlines for the user to commit.

Finally, *wires* connect the components and external interfaces of a module. In the case of *PROCUREMENT* these are *CS*, *SS, SW* and *SC*. Wires are labelled by connectors that coordinate the interactions in which the parties are jointly involved. In SRML, we model the interaction protocols involved in these connectors as separate, reusable entities. Just like business roles and protocols, an interaction protocol is specified in

terms of a number of interactions. Because interaction protocols establish a relationship between two parties, the interactions in which they are involved are divided in two subsets called roles – *A* and *B*. The "semantics" of the protocol is provided through a collection of sentences – what we call *interaction glue* – that establish how the interactions are coordinated. This may include routing events and transforming sent data to the format expected by the receiver. As an example, consider the following protocol used in the wire *SS* that connects *Supplier* and *Stock*:

```
INTERACTION PROTOCOL Custom1 is
   ROLE A
      ask S1(product,nat):bool
      tll S2(product,nat)
      tll S3(product,nat)
   ROLE B
      rpl R1(product):nat
      prf R2(product,nat)
   COORDINATION
      S1(p,n) = R1(p)≥n
      S2(p,n) ⊃ R2(p,R1(p)+n)
      R1(p)≥n ∧ S3(p,n) ⊃ R2(p,R1(p)−n)
      R1(p)<n ⊃ ¬S3(p,n)
```

The wire itself is specified in SRML in a tabular form as follows:

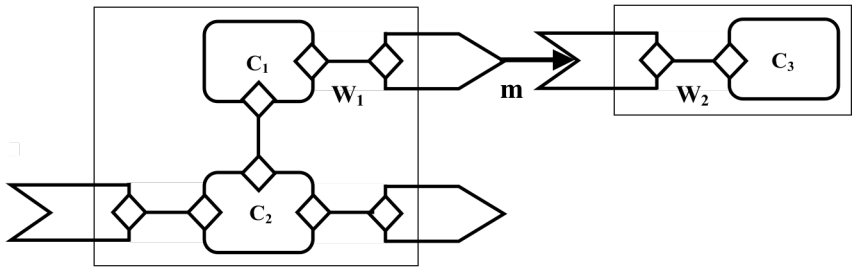
SP Supplier		**SS**		**LS** Stock
ask checkStock	S_1		R_1	**rpl** get
tll incStock	S_2	Custom1	R_2	**prf** set
tll decStock	S_3			

The name bindings instantiate the two roles of the interaction protocol with *Supplier* and *Stock*, respectively, thus establishing that the interactions between the two parties satisfy the following properties:

```
checkStock(p,n)=(get(p)≥n)
incStock(p,n) ⊃ set(p,get(p)+n)
get(p)≥n ∧ decStock(p,n) ⊃ set(p,get(p)−n)
get(p)<n ⊃ ¬decStock(p,n)
```

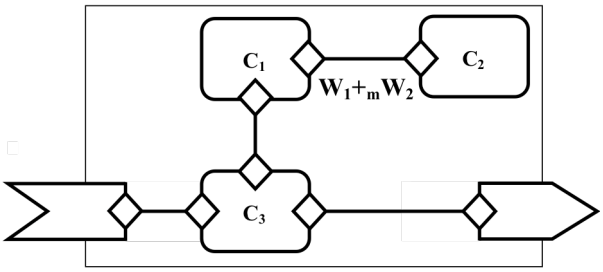
That is, the boolean value returned by *checkStock(p,n)* as invoked by the supplier is computed by the local stock by checking if the value returned by *get(p)* is greater or equal to *n*. The protocol also stipulates that to a request from the supplier for *incStock(p,n)* the local stock executes *set(p,get(p)+n)*. Likewise, to a request from the supplier for *decStock(p,n)* the local stock executes *set(p,get(p)–n)* only if *get(p)* returns a value greater than or equal to *n*; otherwise, the request is not accepted.

The fact that business protocols are specifications over a logic of interactions is important because it will allow us to compose modules by matching the properties required by an external interface of one module with those provided by another. The matching involves what we call an *external wire m*: this is a mapping from the interactions of the "requires" external interface to the interactions of the "provides" external interface that preserves the properties, i.e. *m* defines an interpretation between the theories of the business protocols involved.



An external wire is based on an "empty" interaction protocol, i.e. it does not superpose any additional coordination effects to the (internal) wires W_1 and W_2, it just binds the interactions declared in the external interfaces.

The composition of the two modules results from the composition of the two wires W_1 and W_2 via the mapping m to provide a wire between the two components.



Notice that there is no composition law on components, just on connectors (which extends to wires). Components correspond to software applications, possibly implemented in different languages and running in different platforms; therefore, it does not make sense to compose in the same way that, for instance, a compiler links a number of modules to produce an executable program. This is where we see the difference between social and physiological complexity as already mentioned, which motivates the need for a different algebraic approach.

3 The Algebraic Structure of Connectors

An algebraic formalisation of this notion of module and module composition has been given in [8] from the point of view of a notion of correctness defined based on the theory of institutions [10]. In this paper, we will explore the algebraic structure of connectors in more detail and in a more general setting that does not require the level of detail that we used in [8].

As motivated in Section 2, interactions constitute the core and the unifying element of the proposed approach to systems modelling: all the models that we work with – business roles, business protocols and interaction protocols – are based on structures of interactions. We assume that these structures are organised in a category ***SIGN*** (of signatures) whose morphisms capture "part-of" relationships, i.e. a morphism $\sigma:S_1 \rightarrow S_2$ formalises the way a signature (structure of interactions) S_1 is part of S_2 up to

a possible renaming of the interactions and corresponding parameters. In order to support composition, we further assume that ***SIGN*** is finitely co-complete.

The other structure that is important for interaction protocols is that of the glues; we assume that glues can themselves be organised in a category ***IGLU*** and that a functor ***sign:IGLU→SIGN*** returns, for every glue, the structure of interactions (signature) that are being coordinated by the protocol. As a consequence, a morphism $\sigma:G_1 \rightarrow G_2$ of glues captures the way G_1 is a sub-protocol of G_2, again up to a possible renaming of the interactions and corresponding parameters. That is, σ identifies the glue that, within G_2, captures the way G_1 coordinates the interactions ***sign***(G_1) as a part of ***sign***(G_2). In fact, because we need to be able to compose interaction protocols, we assume that ***IGLU*** is also a finitely co-complete category.

In this formal setting, every interaction protocol P consists of an interaction glue G together with two signature morphisms $\pi_A:roleA \rightarrow$ ***sign***(G) and $\pi_B:roleB \rightarrow$ ***sign***(G). The fact that the roles of the protocol are signatures, and not glues, is important because, as motivated in Section 2, wires establish interconnections between entities (components or external interfaces) purely through relationships between the interactions in which the entities can be involved. These relationships are "syntactic" and are established through the roles of the interaction protocol. If we were to include properties in the roles, we would be involving the computational properties of the entities to which the role is connected. More precisely, we remain agnostic as to the nature of the entities that we wish to interconnect. The only assumption that we make is that each such entity n has a defined signature ***sign***(n):***SIGN***.

The need for separating the mechanisms available for coordinating interactions from the computations that entities execute internally suggests that we work with *coordinated categories* [5]; asking ***sign:IGLU→SIGN*** to be coordinated means that:

- ***sign*** is faithful, i.e. ***IGLU*** is concrete over ***SIGN*** in the sense of [1].
- ***sign*** lifts colimits, i.e. given any diagram ***dia:I→IGLU*** and colimit $(\textbf{\textit{sign}}(G_i) \rightarrow A)_{i:\textbf{\textit{I}}}$ of (***dia***;***sign***) there exists a colimit $(G_i \rightarrow G)_{i:\textbf{\textit{I}}}$ of ***dia*** such that ***sign***$(G_i \rightarrow G) = (\textbf{\textit{sign}}(G_i) \rightarrow A)$.
- ***sign*** has discrete structures in the sense of [1], i.e. every signature A has a 'discrete lift' meaning that there exists $iglu(A)$:***IGLU*** such that, for every $f:A \rightarrow$ ***sign***(G), there is $f':iglu(A) \rightarrow G$ such that ***sign***$(f')=f$.

These properties capture the notion of separation of 'coordination' from 'computation' in the following sense:

- Making ***sign*** faithful means that the computational aspects do not give rise to other interactions than those captured through signatures.
- Lifting colimits means that glues can be composed if their signatures can, and that the signature of the composed glue does not depend on the computations performed by the components.
- The existence of discrete structures means that every signature A has a "realisation" (a discrete lift) as a glue $iglu(A)$ in the sense that, using A to interconnect a glue G, which is achieved through a morphism $f:A \rightarrow$ ***sign***(G), is tantamount to using $iglu(A)$ through any $f':iglu(A) \rightarrow G$ *such that* ***sign***$(f')=f$. Notice that, because ***sign*** is faithful, there is only one such f', which means that f and

f' are, essentially, the same. That is, sources of morphisms in diagrams in ***IGLU*** are, essentially, signatures, which is why we decided to work with structured morphisms in interaction protocols.

Coordinated categories have strong algebraic properties, almost as strong as those of topological categories [1]: a coordinated category is topological iff ***sign*** lifts colimits uniquely. Examples include specifications as theories (or theory presentations) in institutions [10], as well as models of concurrency [13] where signatures consist of process alphabets.

Some of the properties that we will find useful are [5]:

- The functor ***sign*** admits a left adjoint ***iglu***:***SIGN***→***IGLU***.
- The units of the adjunction are identities and the co-units are epis.
- The functor ***sign*** preserves colimits.

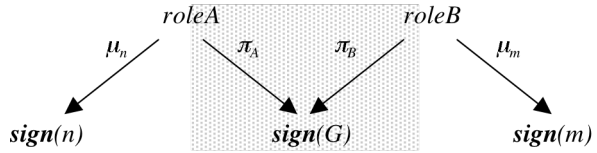
In order to understand the role played by interaction protocols, consider once again the wire *SS* discussed in Section 2:

SP Supplier		**SS**		**LS** Stock
ask checkStock	S_1		R_1	**rpl** get
tll incStock	S_2	Custom1	R_2	**prf** set
tll decStock	S_3			

The wire establishes two signature morphisms: one from the *ROLE_A* of *Custom1* to the signature of *Supplier*, and the other from *ROLE_B* to *Stock*. For instance, the latter is given by the following fragment of the table:

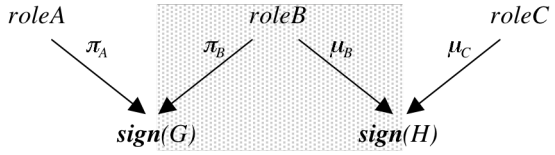
ROLE_B	μ	Stock
R_1	→	get
R_2	→	set

We call a *connector* for a wire $n \leftrightarrow m$ between entities n and m in a module, a structure $<\mu_n,\pi_A,G,\pi_B,\mu_m>$ where $<\pi_A,G,\pi_B>$ is an interaction protocol P and $<\mu_n,\mu_m>$ are the morphisms that connect the roles of P to the entities n and m. Such a connector defines the following diagram in ***SIGN***:



The interaction protocol $<\pi_A,G,\pi_B>$ corresponds to the shadowed part of the diagram. Although this fragment is a co-span in ***SIGN***, the protocol itself is not because it involves the glue G. Indeed, without the computational aspects of the glue it would not be possible to coordinate the interactions between n and m. That is, co-spans in ***SIGN*** are not expressive enough to formalise interaction protocols.

The significance of the difference becomes apparent when we consider the composition of two interaction protocols $<\pi_A,G,\pi_B>$ and $<\mu_B,H,\mu_C>$. We know how to compose the corresponding co-spans in ***SIGN*** through a pushout of the shadowed triangle, but the pushout does not deliver us a glue:



On the other hand, we have already seen that working with co-spans in ***IGLU*** does not make sense because, by allowing the roles to involve computational aspects, the morphisms that connect the roles of the protocol to the entities would bring in computational aspects of the entities into their interconnection.

This is the motivation for studying the properties of structures of the form $<\pi_A,G,\pi_B>$, which we call *structured co-spans*. More precisely, our aim is to define and study the properties of a bicategory whose objects are signatures and whose 1-cells consist of interaction protocols.

4 Structured Co-spans

In this section we define and study the algebraic properties of structured co-spans. We start by recalling some basic definitions and properties of bicategories but only as a reminder – we refer the reader to either the original paper by Bénabou [2] or the more accessible textbook [3]; notice that many other papers are available on this topic but the terminology may change slightly from the one that we use ([3]).

We start by recalling that bicategories were introduced to consider generalisations of categorical constructions to the case in which the identity and composition laws are satisfied only "up to isomorphism". A *bicategory* $\boldsymbol{V}$ consists of:

- A class $|\boldsymbol{V}|$ of objects (also called 0-cells)
- For each pair $<A,B>$ of objects, a category $\boldsymbol{V}(A,B)$ whose objects are called arrows (or 1-cells) and whose morphisms are called 2-cells
- For every triple $<A,B,C>$ of objects, a composition law given by a (bi)functor $;_{A,B,C}: \boldsymbol{V}(A,B)\times\boldsymbol{V}(B,C)\rightarrow\boldsymbol{V}(A,C)$
- For every object A an identity arrow $1_A:A\rightarrow A$

The typical axioms of categories are replaced by the existence of a number of natural isomorphisms and coherence conditions. For simplicity, we omit these properties and refer the reader to [3]. We have already mentioned that typical examples of bicategories in computer science are (co-)spans of graphs [11,14].

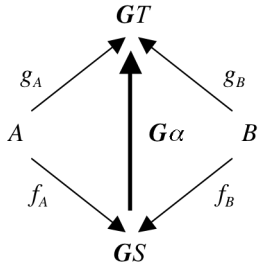
A similar generalisation applies to functors. Given bicategories $\boldsymbol{V}$ and $\boldsymbol{W}$, a *lax functor* $\boldsymbol{F}:\boldsymbol{V}\rightarrow\boldsymbol{W}$ consists of:

- A map sending objects A of $\boldsymbol{V}$ to objects $\boldsymbol{F}A$ of $\boldsymbol{W}$
- Functors $\boldsymbol{F}_{A,B}:\boldsymbol{V}(A,B)\rightarrow\boldsymbol{W}(\boldsymbol{F}A,\boldsymbol{F}B)$ for every pair $<A,B>$ of objects of $\boldsymbol{V}$, a
- 2-cells $\boldsymbol{F}^{\boldsymbol{2}}_{f,g}:\boldsymbol{F}f;\boldsymbol{F}g\rightarrow\boldsymbol{F}(f;g)$ for every composable $<f,g>$ in $\boldsymbol{V}$, natural in f and g
- 1-cells $\boldsymbol{F}^{\boldsymbol{0}}_{A}:1_{\boldsymbol{F}A}\rightarrow\boldsymbol{F}1_A$ for every object A of $\boldsymbol{V}$

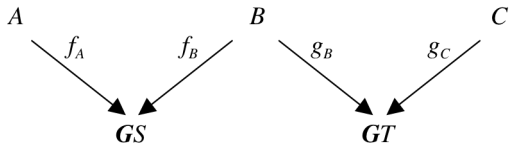
subject to coherence conditions [3]. A lax functor $\boldsymbol{F}$ is a *pseudo-functor* when all the $\boldsymbol{F}^{\boldsymbol{2}}_{f,g}$ and $\boldsymbol{F}^{\boldsymbol{0}}_{A}$ are invertible.

Definition 4.1: Given an adjunction $\boldsymbol{F} \dashv \boldsymbol{G}:\boldsymbol{D} \to \boldsymbol{C}$, where $\boldsymbol{D}$ has pushouts, we define the bicategory ***co-span(G)*** of $\boldsymbol{G}$-structured co-spans as follows:

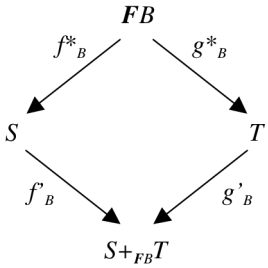
- The objects are those of $\boldsymbol{C}$
- The arrows (1-cells) are triples $\langle f_A : A \to \boldsymbol{G}S, S, f_B : B \to \boldsymbol{G}S \rangle$ where S is an object of $\boldsymbol{D}$ and f_A, f_B are morphisms of $\boldsymbol{C}$
- A 2-cell $\alpha : \langle f_A, S, f_B \rangle \to \langle g_A, T, g_B \rangle$ is a $\boldsymbol{D}$-morphism $\alpha : S \to T$ that makes the following diagram commute



- Composition of $\langle f_A, S, f_B \rangle$ and $\langle g_B, T, g_C \rangle$



is $\langle f_A ; \boldsymbol{G}f'_B, S +_B T, g_C ; \boldsymbol{G}g'_B \rangle$ obtained through the following pushout in $\boldsymbol{D}$ where $f^*_B = \boldsymbol{F}f_B ; \varepsilon_S$ and $g^*_B = \boldsymbol{F}g_B ; \varepsilon_T$



- The identities are $\langle id_A ; \eta_A, \boldsymbol{F}A, id_A ; \eta_A \rangle$.

As could be expected:

Remark 4.2: For every category $\boldsymbol{C}$ with pushouts, ***co-span***(1_C) is the well-know category of co-spans over $\boldsymbol{C}$, which we denote by ***co-span(C)***.

The following property is easily proved:

Proposition 4.3: Let $\boldsymbol{F} : \boldsymbol{C} \to \boldsymbol{D}$ be a functor between two categories with pushouts.

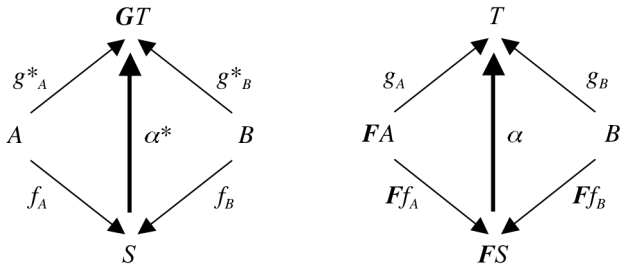
- $\boldsymbol{F}$ extends to a lax $\dot{\boldsymbol{F}}$***:co-span(C)*** $\to$ ***co-span(D)*** by pointwise translation.
- $\dot{\boldsymbol{F}}$ is normal, i.e. it sends identity 1-cells to identities.
- If $\boldsymbol{F}$ preserves pushouts, $\boldsymbol{F}$ is a pseudo-functor.

We are now going to analyse the lifting of adjunctions. As established in [12], bi-categories admit more general morphisms than lax functors – the so-called "2-sided enrichments", which together with the appropriate 2-cells and 3-cells form the so-called tricategory ***Caten***. The definition of adjoint one-cells makes sense in any bi-category and, in particular, in ***Caten*** where they are characterised as follows:

Theorem 4.4 ([12] Proposition 2.7): A left adjoint 2-sided enrichment $\boldsymbol{F:V\to W}$ is a pseudo-functor such that each functor $\boldsymbol{F_{A,B}: V(A,B)\to W(FA,FB)}$ has a right adjoint.

Consider now the case in which we are given an adjunction $\boldsymbol{F\dashv G:D\to C}$, where ***D*** and ***C*** have pushouts:

- Because ***F*** preserves pushouts, $\dot{\boldsymbol{F}}$ is a pseudo-functor
- Each functor $\dot{\boldsymbol{F}}_{A,B}$: ***co-span(C)(A,B)→co-span(D)***$(\dot{\boldsymbol{F}}A,\dot{\boldsymbol{F}}B)$ has a right-adjoint based on the isomorphisms between the two hom-sets:

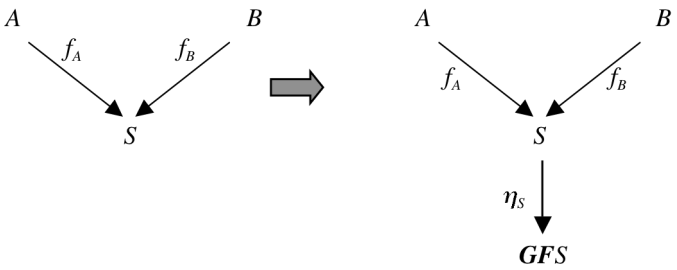


Corollary 4.5: Given an adjunction $\boldsymbol{F\dashv G:D\to C}$ where ***D*** and ***C*** have pushouts, $\dot{\boldsymbol{F}}$***:co-span(C)→co-span(D)*** is a left adjoint 2-sided enrichment.

Notice that nothing can be inferred from this result about the lax functor $\dot{\boldsymbol{G}}$***:co-span(D)→co-span(C)***. We are now going to see that ***co-span(G)*** allows us to strengthen the case.

Proposition 4.6: Given an adjunction $\boldsymbol{F\dashv G:D\to C}$ where ***D*** and ***C*** have pushouts, we define a pseudo functor $\boldsymbol{F^*:co\text{-}span(C)\to co\text{-}span(G)}$ as follows:

- $\boldsymbol{F^*}$ is the identity on objects
- Every 1-cell $\langle f_A,S,f_B\rangle$ is mapped to $\langle f_A;\eta_S,\boldsymbol{FS},f_B;\eta_s\rangle$, and the 2-cells $\alpha:\langle f_A,S,f_B\rangle\to\langle g_A,T,g_B\rangle$ to $\boldsymbol{F}\alpha:\langle f_A;\eta_S,\boldsymbol{FS},f_B;\eta_S\rangle\to\langle g_A;\eta_T,\boldsymbol{FT},g_B;\eta_S T\rangle$. Notice that, being a left adjoint, ***F*** preserves colimits, which justifies that we do obtain a pseudo functor.



If we consider the hom-categories, it is easy to see that we have lifted the adjunction $\boldsymbol{F} \dashv \boldsymbol{G}:\boldsymbol{D}\rightarrow\boldsymbol{C}$ to an adjunction $\boldsymbol{co\text{-}span(G)}(A,B)\rightarrow\boldsymbol{co\text{-}span(C)}(A,B)$.

Proposition 4.7: $\boldsymbol{F}^*:\boldsymbol{co\text{-}span(C)}\rightarrow\boldsymbol{co\text{-}span(G)}$ is a left adjoint 2-sided enrichment. Moreover, because $\boldsymbol{F}^*$ is the identity on objects, we obtain a right adjoint that is a lax functor $^*\boldsymbol{G}:\boldsymbol{co\text{-}span(G)}\rightarrow\boldsymbol{co\text{-}span(C)}$.

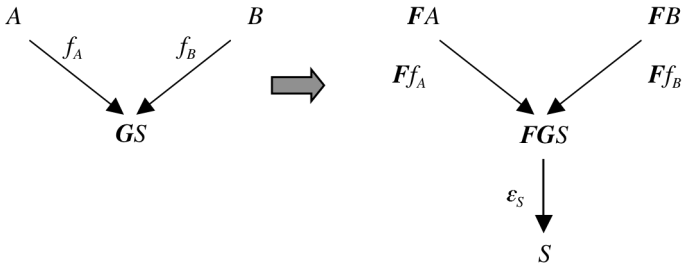
It is interesting to analyse the construction of the right-adjoint:

- $^*\boldsymbol{G}$ is again the identity on objects
- Every 1-cell $\langle f_A,S,f_B\rangle$ is mapped to $\langle f_A,\boldsymbol{G}S,f_B\rangle$, and the 2-cells $\alpha:\langle f_A,S,f_B\rangle\rightarrow\langle g_A,T,g_B\rangle$ to $\boldsymbol{G}\alpha:\langle f_A,\boldsymbol{G}S,f_B\rangle\rightarrow\langle g_A,\boldsymbol{G}T,g_B\rangle$.

Recall that the identities for structured co-spans are of the form $\langle id_A;\eta_A,\boldsymbol{F}A,id_A;\eta_A\rangle$. Hence, $^*\boldsymbol{G}_A=\eta_A$. Moreover, $\boldsymbol{G}$ does not necessarily preserve pushouts. This is why we cannot guarantee that $^*\boldsymbol{G}$ is a pseudo-functor. However if $\boldsymbol{G}$ defines a coordinated category, we know that it preserves colimits and the units of the adjunction are identities.

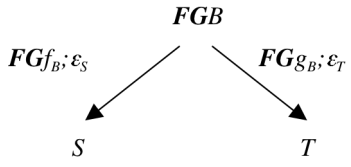
Proposition 4.8: If $\boldsymbol{F} \dashv \boldsymbol{G}:\boldsymbol{D}\rightarrow\boldsymbol{C}$ is a coordinated category, we have an adjunction $\boldsymbol{F}^* \dashv {}^*\boldsymbol{G}:\boldsymbol{co\text{-}span(G)}\rightarrow\boldsymbol{co\text{-}span(C)}$ of pseudo functors.

Consider now what happens on the side of $\boldsymbol{co\text{-}span(D)}$. We have an obvious pseudo functor based on $\boldsymbol{F}$ and the functors $\boldsymbol{co\text{-}span(G)}(A,B)\rightarrow\boldsymbol{co\text{-}span(D)}(\boldsymbol{F}A,\boldsymbol{F}B)$ that map structured co-spans $\langle f_A,S,f_B\rangle$ to the co-spans $\langle f^*_A,S,f^*_B\rangle$:

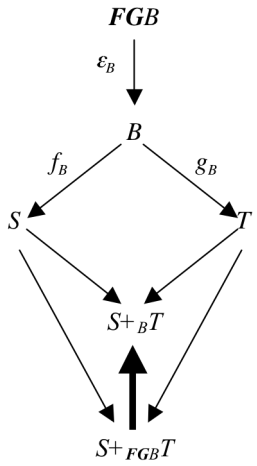


These functors are isomorphisms, leading to a left adjoint 2-sided enrichment $^*\boldsymbol{F}$.

Mapping co-spans $\langle f_A,S,f_B\rangle$ over $\boldsymbol{D}$ to structured co-spans $\langle \boldsymbol{G}f_A,S,\boldsymbol{G}f_B\rangle$ seems equally obvious, but the mapping of the composition deserves some attention. If we consider $\langle f_A,S,f_B\rangle;\langle g_B,T,g_C\rangle$, the composition of the images is given by a pushout of:



Because $\boldsymbol{FG}f_B;\varepsilon_S=\varepsilon_B;f_B$ and $\boldsymbol{FG}g_B;\varepsilon_T=\varepsilon_B;g_B$, we have in fact:



The universal properties of the colimit return a morphism $S+_{FGB}T \to S+_BT$. If we work with a coordinated category, $\boldsymbol{G}$ is faithful, which implies that the co-units are epis. In this case, it is easy to see that the morphisms $S+_{FGB}T \to S+_BT$ are in fact isomorphisms, which makes $\boldsymbol{G^*}$ a pseudo functor.

We can now summarise our results.

Theorem 4.9: Let $\boldsymbol{F:C \to D}$ be a functor between two categories with pushouts.

- $\boldsymbol{F}$ extends to a normal lax functor $\boldsymbol{\dot{F}:co\text{-}span(C) \to co\text{-}span(D)}$; if $\boldsymbol{F}$ preserves pushouts, $\boldsymbol{\dot{F}}$ is a pseudo-functor.
- If $\boldsymbol{F}$ has a right adjoint $\boldsymbol{G:D \to C}$:
- $\boldsymbol{\dot{F}}$ is a left adjoint 2-sided enrichment.
- $\boldsymbol{\dot{F}}$ factorises as $co\text{-}span(C) \xrightarrow{F^*} co\text{-}span(G) \xrightarrow{^*F} co\text{-}span(D)$ where $\boldsymbol{F^*}$ has a lax right adjoint $\boldsymbol{^*G}$

Theorem 4.10: Let $\boldsymbol{G:D \to C}$ be a coordinated category.

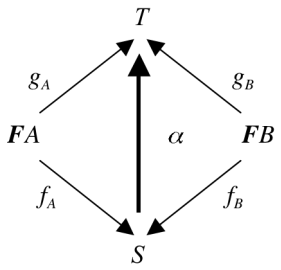
- The right adjoint $\boldsymbol{^*G}$ is a pseudo functor
- $\boldsymbol{^*F}$ has a pseudo right adjoint $\boldsymbol{G^*}$
- $\boldsymbol{\dot{F} \dashv \dot{G}:co\text{-}span(D) \to co\text{-}span(C)}$ is an adjunction of pseudo-functors

Our final result is a generalisation of the factorisation that we defined for $\boldsymbol{\dot{F}:co\text{-}span(C) \to co\text{-}span(D)}$ to a general lax functor.

Definition 4.11: Given a lax functor $\boldsymbol{F:V \to W}$, we define the bicategory $\boldsymbol{V_F}$ as follows:

- $\boldsymbol{|V_F| = |V|}$
- $\boldsymbol{V_F(A,B) = W(FA,FB)}$

Notice that, in the case of $\boldsymbol{\dot{F}:co\text{-}span(C) \to co\text{-}span(D)}$ what we obtain is a bicategory whose hom-cats are of the form:



which are isomorphic to ***co-span(G)(A,B)*** if ***F*** has a right adjoint ***G:D→C***.

Our last result is a canonical factorisation of left adjoint 2-sided enrichments:

Theorem 4.12: Every lax functor ***F:V→W*** factorises as $V \xrightarrow{F^*} V_F \xrightarrow{{}^*F} W$ where ***F**** is an identity on objects and ****F*** an identity on hom-cats. If ***F*** is a left adjoint 2-sided enrichment so is ***F**** and its right adjoint is lax.

5 Concluding Remarks

In this paper, we have shown how the notion of interaction protocol that we are developing within the SENSORIA project used for modelling interconnections in service-oriented systems can be given an algebraic semantics over an extension of the theory of co-spans. The extension is motivated by the fact that, whereas we want the interaction protocol to use a rich formalism to specify the coordination mechanisms superposed by the glue, its interfaces should be purely "syntactic" so as to avoid any assumption on the computations performed by the entities being interconnected.

More precisely, given a coordinated category ***sign:IGLU→SIGN***, using ***co-span(SIGN)*** for interconnections is too poor because it does not support the definition of coordination mechanisms, and using ***co-span(IGLU)*** is too strong because the interfaces involve computational aspects. This is why we proposed to work over an algebraic structure ***co-span(sign)*** that is based instead on ***sign***-structured morphisms.

We showed how ***co-span(sign)*** constitutes a bicategory. In fact, we investigated the more general issue of how the ***co-span*** construction relates to functors. We showed how a functor between the base categories induces a lax-functor between the corresponding bicategories of co-spans, and how adjunctions give rise to adjoint 2-sided enrichments. This allowed us to strengthen some results on adjunctions in the tricategory ***Caten***, namely by generalising the construction of ***co-span(sign)*** to a canonical factorisation of lax functors. This is a line that we would like to pursue on its own, although the "computational" inspiration that comes from (structured) co-spans and coordinated categories is very welcome.

From the point of view of software-intensive system modelling, it is clear that structured morphisms over coordinated categories have been proving to provide a richer algebraic framework when it comes to formalising interconnection mechanisms. This is another avenue that we want to keep exploring in SENSORIA.

Acknowledgments

We would like to acknowledge the contribution of Antónia Lopes with whom much of the work around coordinated categories in general, and interaction protocols in particular, has been developed, and to thank our colleagues in SENSORIA, the IFIP WG1.3 group members and observers, and the participants of the Workshop on Applied and Computational Category Theory (ACCAT) 2007 for valuable feedback.

References

1. J. Adámek, H. Herrlich, G. Strecker (1990) *Abstract and Concrete Categories*. John Wiley & Sons, New York Chichester Brisbane Toronto Singapore
2. J. Bénabou (1967) Introduction to bicategories. In: *Midwest Category Seminar. LNCS, vol 42*. Springer, Berlin Heidelberg New York,pp 1–77
3. F. Borceux (1994) *Handbook of Categorical Algebra 1*. Cambridge University Press, Cambridge
4. H. Ehrig, F. Orejas, B. Braatz, M. Klein, M. Piirainen (2004) A component framework for system modeling based on high-level replacement systems. *Software Systems Modeling* 3:114–135
5. J. L. Fiadeiro (2004) *Categories for Software Engineering*. Springer, Berlin Heidelberg New York
6. J. L. Fiadeiro (2007) Designing for software's social complexity. *IEEE Computer* 40(1):34–39
7. J. L. Fiadeiro, A. Lopes, L. Bocchi (2006) A formal approach to service-oriented architecture. In: M. Bravetti, M. Nunez, G. Zavattaro (eds) *Web Services and Formal Methods. LNCS, vol 4184*. Springer, Berlin Heidelberg New York, pp 193–213
8. J. L. Fiadeiro, A. Lopes, L. Bocchi (2007) Algebraic semantics of service component modules. In: J. L. Fiadeiro, P. Y. Schobbens (eds) *Algebraic Development Techniques. LNCS, vol 4409*. Springer, Berlin Heidelberg New York, pp 37–55
9. J. L. Fiadeiro, A. Lopes, M. Wermelinger (2003) A mathematical semantics for architectural connectors. In: R. Backhouse, J. Gibbons (eds) *Generic Programming. LNCS, vol 2793*. Springer, Berlin Heidelberg New York, pp 190–234
10. J. Goguen, R. Burstall (1992) Institutions: abstract model theory for specification and programming. *Journal ACM* 39(1):95–146
11. P. Katis, N. Sabadini, R. F. C. Walters (1997) Bicategories of processes. *Journal of Pure and Applied Algebra* 115:141–178
12. G. M. Kelly, A. Labella, V. Schmitt, R. Street (2002) Categories enriched on two sides. *Journal of Pure and Applied Algebra* 168:53–98
13. V. Sassone, M. Nielsen, G. Winskel (1993) A classification of models for concurrency. In: E. Best (ed) *CONCUR'93. LNCS, vol* 7159. Springer, Berlin Heidelberg New York, pp 82-96
14. V. Sassone, P. Sobocinski (2005) Reactive systems over cospans. *LICS'05*, IEEE Computer Society, pp 311–320