Using Motivation and Choreography to model Distributed Workflow

Ashley McNeile
Metamaxim Ltd.
48 Brunswick Gardens
London, U.K.
ashley.mcneile@metamaxim.com

ABSTRACT

We describe an approach to modeling the design of distributed interactive workflow based on the combining the ideas of modeling *motivation* and *choreography*.

We use the concept of motivation to differentiate between actions that are *solicited* by the workflow and actions that are not, and happen *spontaneously*. We recap on a previously described analysis technique that can verify that a local (non-distributed) workflow modeled in this way is bound to progress to a business completion in finite time.

We use a choreography to model distributed behavior where multiple participants communicate using asynchronous message exchange. We show how choreography and motivation can be combined in a single model to represent a distributed colllaborative workflow. We show that the analysis technique for guaranteed completion can be extended to distributed workflow by applying it separately to each participant and to the choreography.

We compare our work with other approaches to modeling workflow, and show that modeling motivation supports better analysis of progress and completion where actions can both be solicited and happen spontaneously.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*model checking*; D.2.10 [**Software Engineering**]: Design—*Representation*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Modal Logic*; H.4.1 [**Information Systems Applications**]: Office Automation—*Workflow management*

General Terms

Design, Theory, Verification

Keywords

Interactive Workflow, Model Checking, Verification, Process Algebra, Choreography

BM-FA '13 July 2, 2013, Montpellier, France

1. INTRODUCTION

The use of workflow to help manage and drive business processes has been around for over twenty years. In the early days such systems were seen as electronic replacements for paper-based workflow systems but in recent times, driven by a combination of increasing service specialization and ubiquitous network connectivity both inside and across organizations, they are enabling new business models based around service orientation and business process outsourcing. This is now manifesting itself as business process integration across organization boundaries in such areas as supply chain, order processing, claims handling and financial trading.

When one organization relies on others for parts of its business process it commonly specifies performance and quality targets. Such targets are increasingly being formalized in legal Service Level Agreements (SLAs) carrying financial penalties for non-compliance, and so it is becoming a commercial imperative to ensure that workflow automation is designed to meet end-to-end service time targets under all possible scenarios. This paper describes an analysis technique that can be used to ensure that a proposed workflow design will meet the intended service time performance, even when the workflow is distributed between multiple participants.

In particular, the techniques described in this paper address the challenge of modeling and analyzing distributed workflows to ensure adherence to service time requirements where:

- The workflow propels itself forward by soliciting actions from its environment.
- Not all the actions that take place in the workflow are solicited, and some take place spontaneously at the whim of the environment, without solicitation.
- The worklow is distributed across multiple participants who communicate using asynchronous message exchange.

Modeling the distinction between *solicited* and *spontaneous* actions in workflow, and using this to establish that a workflow model employing both solicited and spontaneous actions is *bound to complete*, has been described by McNeile and Roubtsova [2], but only addressing local (non-distributed) workflow. The central contribution of the current paper is to extend the modeling and analysis approach described there to *distributed workflow*, using *choreography* techniques to formalize and verify correctness in distributed behavior.

This paper is organized as follows:

Section 2 explains the idea of *motivation* and the difference between solicited and spontaneous actions in workflow.

Section 3 shows how motivation in workflow is modeled and illustrates this with a small example.

Section 4 discusses how a workflow is analyzed to show that it completes.

Section 5 shows how the ideas in Section 3 are combined with *choreography* techniques to model distributed workflow.

Section 6 shows how the analysis technique described in Section 4 is applied to distributed workflow.

Section 7 relates our work to other research into workflow and modal modeling.

2. WORKFLOW AND MOTIVATION

This paper is concerned with a particular class of process support systems, generally termed "workflow", where the software plays an active role in soliciting actions from the environment. To introduce the idea, we use the example of a car fuel gauge, as shown in Figure 1. The gauge shows the amount of fuel left in the tank and has a *low-fuel light*, shown by the "E" on the left hand side, that comes on when the tank is almost empty: say just one liter of fuel left. Similarly a business workflow is able to determine what actions should happen next and then interact with the environment to advance the process.

2.1 Solicited and Spontaneous Actions

When the low-fuel light is lit we talk of refueling being *solicited*. Of course, the driver is quite free to refuel the car when the light is not lit, and then we talk of refueling being *spontaneous*. Similarly in a workflow we have both solicited and spontaneous actions. For instance, in an order processing workflow operated by a supplier, a customer places an order and as a result it appears on an electronic "to-do list" of an employee at the supplier who is prompted by the workflow system to perform a credit check and arrange a delivery date. These actions are solicited, or prompted, by the workflow. However the customer who placed the order may choose to amend or cancel it, perhaps entering their amendment or cancelation to the supplier's system via a web based-interface. Such an action would obviously not be solicited by the workflow and would therefore, in our terminology, be modeled as spontaneous. Spontaneous actions in a workflow are not performed in response to an electronic to-do list and are, in this sense, unexpected.

2.2 Operator Commitments

In order for the low-fuel light to be effective, the operator (the driver of the car) has to understand its meaning. Specifically, we assume that the driver agrees to observe an *Operator Commitment* (hereafter abbreviated **OC**) in this case as follows: *As soon as the light comes on, the driver undertakes that within 20 km the car is refueled sufficiently for the light to go off again.* This OC gives a behavioral meaning to the light and, as along as it is obeyed, the car will never run out of fuel (provided that a litre of fuel is at least enough for 20 km of travel). As in the case of the low-fuel light, the operators from whom action is solicited by a workflow are bound by agreed OCs to react in a timely manner. In the business world such an agreement would normally be part of a formal (in the sense of agreed and documented) service level agreement (SLA). We have chosen to use the generic term *Operator Commitment* instead of SLA as these encompass aspects of service management and quality (such as reliability, availability and cost) that we do not consider.

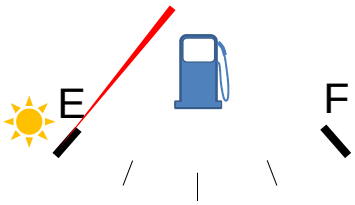

Figure 1: Car Fuel Gauge

We will assume that, when an action is solicited from an agent by a workflow, the agent is subject to an OC under which the agent undertakes to complete the action within a defined time. We will use this to reason about the time required for a workflow to complete.

3. MODELING WORKFLOW

This section provides a summary of the approach to modeling workflow given in [2].

3.1 Can Model and Want Model

To model the distinction between solicited and spontaneous actions we create two separate models, which we call the *can-model* and the *want-model*. At any given state of the workflow, the can-model states what actions *can happen*, whether solicited or not. Similarly, at any given state of the workflow, the want-model states what actions *need to happen* to achieve progress. The want-model defines which actions the workflow solicits from external agents by causing appropriate entries to appear on their to-do lists. We say that the want-model provides the *motivation* in the workflow.

We use a state-transition based modeling technique called *Protocol Modeling* (hereafter abbreviated **PM**) as described McNeile et al. in [3] to represent both the can- and the want-models of a workflow, as illustrated in the next section.

3.2 Example Workflow

Figure 2 shows a simple expense claim approval process. The process involves actions by an **Employee**, who may *Submit* and *Withdraw* a claim; a **Supervisor**, who reviews a claim and can *Pass* or *Fail* the claim; and a **Manager** who may *Authorize* payment against claim once it has been passed by the Supervisor. The Manager has discretion to authorize less than the full claim amount. If the Supervisor does not pass the claim on review, the Employee may *Correct* and *Resubmit* it after correction, but may only resubmit once.

Figure 2 is the *can-model* of the workflow, and is expressed as the composition of two conventional state machines: $EC1 \parallel EC2$. The first machine, $EC1$, shows the main path of the process. The second, $EC2$, represents the

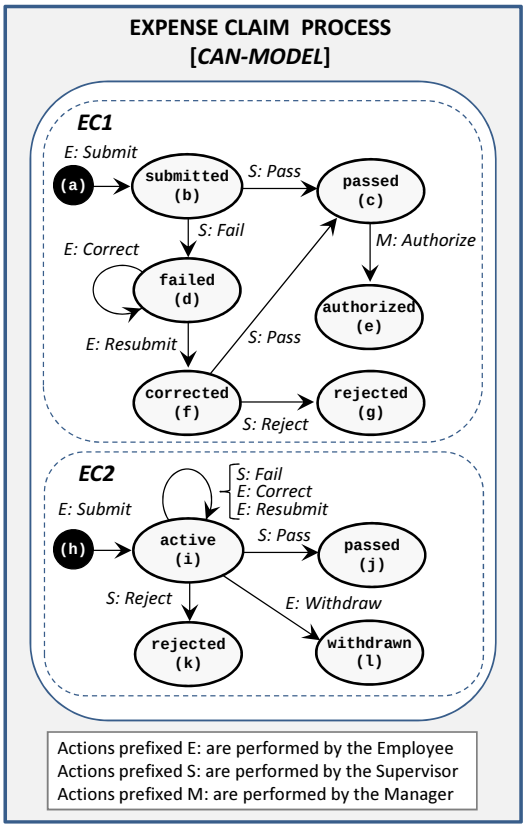

Figure 2: Expense Claim Process: Can-Model

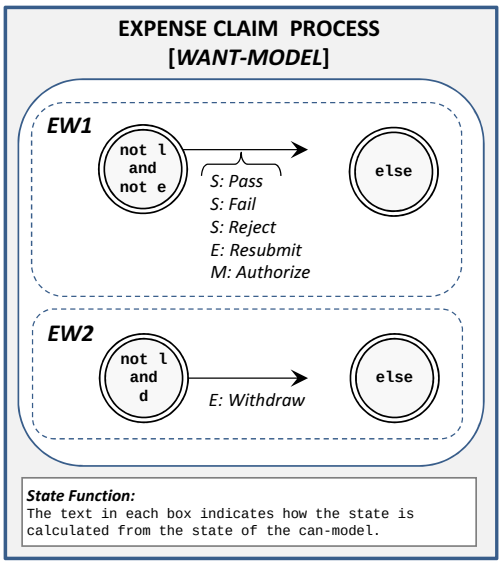

Figure 3: Expense Claim Process: Want-Model

fact that the Employee may *Withdraw* the claim at any point up to when it is passed or rejected.

However, this can-model by itself is like the fuel gauge without the low-fuel light: there is no model of motivation. Figure 3 shows the *want-model* for the expense claim workflow and models the motivation, by showing when actions

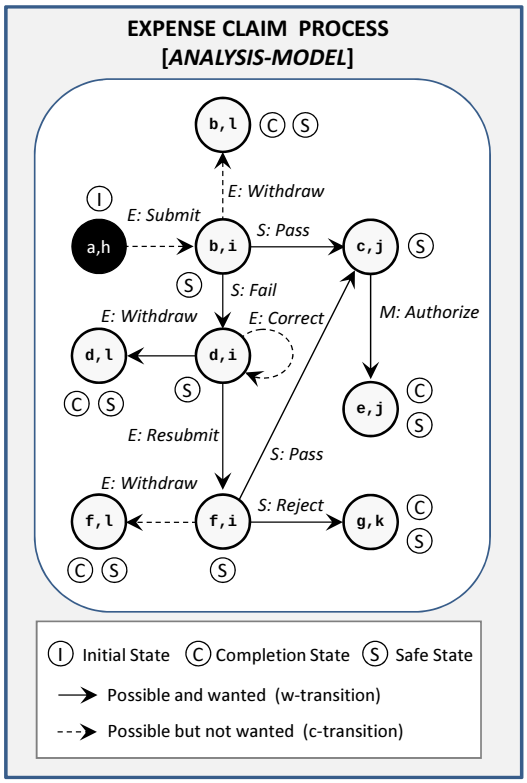

Figure 4: Expense Claim Process: Analysis-Model

are solicited. The want-model consists of two composed derived-state machines: $EW1 \parallel EW2$ (although, as their alphabets are disjoint, these machines are effectively independent). Both machines use a state derived from the state of the can-model so that, for instance, when $EC1$ is in state `submitted` (which has label `(b)`) and $EC2$ is *not* in state `withdrawn` (label `(l)`) then the want-model indicates that *Pass* and *Fail* are both "wanted actions". Although both of these actions are wanted, the can-model ensures that *only one of them* can actually take place.

4. PROGRESS AND COMPLETION ANALYSIS

Our aim is to construct workflows that are bound to progress. Progress analysis is based on examination of the topology of the can- and want-models. The key idea is that progress relies on the occurrence of actions that are solicited by the want-model, but must not be jeopardized by occurrence of spontaneous actions.

4.1 Analysis Model

The basis for analyzing the workflow for progress is to re-express the can-model as a single consolidated machine. This re-expression uses a standard technique, described in [2]. The want-model can then be used to distinguish between different types of transition in the consolidated can-model as follows:

- Where a transition is possible (allowed by the can-model) *but not* wanted (by the want-model) it is termed a *c-transition*.

- Where a transition is possible (allowed by the can-model) *and* wanted (by the want model) it is termed a *w-transition*.

The resultant model is called the *analysis-model*. Figure 4 shows the analysis-model of the expense claim workflow, and shows the w-transitions as solid arrows and the c-transitions as dashed arrows. The analysis of progress and completion of a workflow is performed using this model.

It is important to understand that the role of the analysis-model is *only* to support reasoning about progress and it does *not* usurp the role of being the design of the workflow; this role remains with the individual can- and want-models (such as those shown in Figures 2 and 3).

The ability to classify every transition of the analysis model as either a w-transition or a c-transitions requires that the state of want-model is single valued (unique determined) by the state of the can-model, and therefore that states of a want-model are Boolean expressions of the states of the can-model. As explained in [2] this is not a limiting assumption and for the purpose of this paper we assume it is true.

4.2 Conditions for Progress

As a basis for progress analysis we classify the states in the analysis-model. Suppose that the set of states of the analysis-model is denoted by Σ. We distinguish different kinds of state in Σ as follows.

Initial State. A state $\sigma \in \Sigma$ is an *Initial State* if and only if it is not the end state of any transition (either a c-transition or a w-transition).

Completion State. A state $\sigma \in \Sigma$ is a *Completion State* if and only if no further action is wanted, whether it is possible (allowed by the can-model) or not.

Safe State. A state $\sigma \in \Sigma$ is a *Safe State* if and only if it is either a Completion State, or it is the start state of at least one w-transition and every w-transition from σ leads to a Safe State.

Note that this is not a partitioning of the states of Σ: there may be states that do not qualify as any of these, and states that qualify as more than one type (see Figure 4). Note also that there is no requirement that a Completion State has no outgoing transitions, only that it has no *wanted* outgoing transition.

Based on the above, we can state a sufficient condition for a workflow that starts at an Initial State and is "kicked-off" by a c-transition to maintain progress: *Every c-transition in the analysis-model of the workflow ends at a Safe State.* A short argument shows that this guarantees that the workflow will not rest indefinitely in a given state.

Suppose that the workflow is in a Safe State that is not a Completion State. If an action takes place, it must either be a c-transition or a w-transition. By virtue of the condition given above, a c-transition will take the workflow to a Safe State. By virtue of the definition of Safe State, a w-transition will also take the workflow to a Safe State. Moreover, by well-formedness condition 2 in Section 4.3, the first ("kick-off") transition must be a c-transition and so take the workflow to a Safe State. This means that once the workflow has started it can *never leave* the subgraph of Safe States. Because the time that a workflow can remain in a Safe State that is not a Completion State is bounded by the minimum OC of its wanted outgoing transitions (of which it is required to have at least one), a workflow that is not in a Completion State must progress.

4.3 Well-Formedness

The analysis-model of a properly engineered workflow must have the following properties:

1. It has at least one Initial State and at least one Completion State.

2. Every Initial State has at least one outgoing c-transition and no outgoing w-transition.

3. Two successive w-transitions (one entering a state and the other leaving it) cannot be for the same action.

4. Every Completion State should be a *business completion* of the workflow, in the sense that no party involved expects anything further to happen to complete the business transaction managed by the workflow.

Properties 2 and 3 merit explanation. Property 2, requiring that an Initial State has no outgoing w-transition, ensures that the workflow cannot solicit instances of itself, otherwise the system would flood with instances. Property 3, requiring that we never have two successive w-transitions for the same action, means that once an agent performs a solicited action it disappears from her to-do list, otherwise it is not possible to give a clear definition of an Operator Commitment (OC).

We will assume that every workflow has these four properties.

4.4 Guaranteed Completion

So far we have focused on the conditions for *progress*: ensuring that a workflow will not rest indefinitely in one state. As pointed out in Section 4, the presence of cycles in the model may mean that a workflow may cycle through Safe States without reaching a Completion State. If a well-formed workflow

- obeys the progress rules set out in Section 4, and

- has cycles that are bounded so that all traces are of finite length

then it is described as being **GC** (for Guaranteed Completion). A workflow that is GC is bound to complete in finite time, as described in the next section.

Further discussion on cycles, and methods of establishing that cycling is limited can be found in McNeile and Roubtsova [2].

4.5 Workflow Response Time

Suppose that we have a trace, $\mathcal{T}$, of the workflow which has GC expressed as a sequence $\sigma_i, 0 \leq i \leq n$ of states of the analysis-model, where σ_0 is an Initial State and σ_n is the first encounter of a Completion State. We will now construct the longest time that the workflow can take to follow this trace.

First we form the function $t(i, x)$ for $1 < i < n$ and $x \in \Lambda$

as follows:

$$maxrest(i) = \min_{y \in WT(\sigma_i)} (t(i,y)) \quad (1)$$

$$x \in WT(\sigma_i) \text{ and } x \notin WT(\sigma_{i-1}) \Rightarrow t(i,x) = OC(x) \quad (2)$$

$$x \in WT(\sigma_i) \text{ and } x \in WT(\sigma_{i-1}) \Rightarrow t(i,x) = t(i-1,x) - maxrest(i-1) \quad (3)$$

$$x \notin WT(\sigma_i) \Rightarrow t(i,x) \text{ is undefined.} \quad (4)$$

Here, $t(i,x)$ represents the time that an action x that is possible and wanted in state σ_i has left before breach of its OC occurs; and $maxrest(i)$ is the maximum time that the workflow can remain in state σ_i without violation of any OC. Equation (1) says that $maxrest(i)$ is the minimum of the OC times remaining for any possible and wanted action in state σ_i. Equation (2) says that the time remaining for an action that becomes possible and wanted at the i^{th} step is the OC time of that action. Equation (3) says that the time remaining for an action in that is possible and wanted at the i^{th} step and was also possible and wanted at the $(i-1)^{th}$ step, is reduced by the time that the workflow spends in the $(i-1)^{th}$ step.

Note that $t(i,x)$ can be computed because:

- The GC condition requires that all states for $0 < i < n$ are Safe States with $WT(\sigma_i) \neq \emptyset$.
- By the definition of Initial State, $WT(\sigma_0) = \emptyset$.

The maximum time for the trace $\mathcal{T}$ is then:

$$\sum_{0<i<n} maxrest(i)$$

This can be used to determine whether the end-to-end response time of a workflow conforms to limits that are specified in an SLA (Service Level Agreement) based on the OC limits specified in the various OLAs (Operation-Level Agreement) for the actions involved. In theory, all traces of the workflow would need to be analyzed to determine the maximum response time, which is possible as the absence of cycles means that the number of traces is finite. In practice, a few key paths are important and the analysis does not need to be exhaustive.

5. DISTRIBUTED WORKFLOW

So far this paper provides a somewhat summarized version of the treatment of local (non-distributed) workflow given in McNeile and Roubtsova [2]. The remainder of the paper is new material exploring the possibility that a workflow is distributed across a number of participants connected via a network. We assume that the participants communicate over a network that provides FIFO message transport in both directions between each pair of participants.

5.1 Choreography

Consider the collaboration shown in Figure 5 in which distributed participants {$P1$, $P2$, $P3$} work together. At some point in the collaboration, $P1$ may ask a *Question* [Q] which it sends to $P2$, and we assume that $P1$'s completion is *dependent* on receiving an *Answer* [A], unless $P1$ decides to *Cancel* [C]. If $P2$ does not know the answer then she may pass the question on to $P3$ who replies directly back to $P1$. It is somewhat like a "first and second level support" scenario.

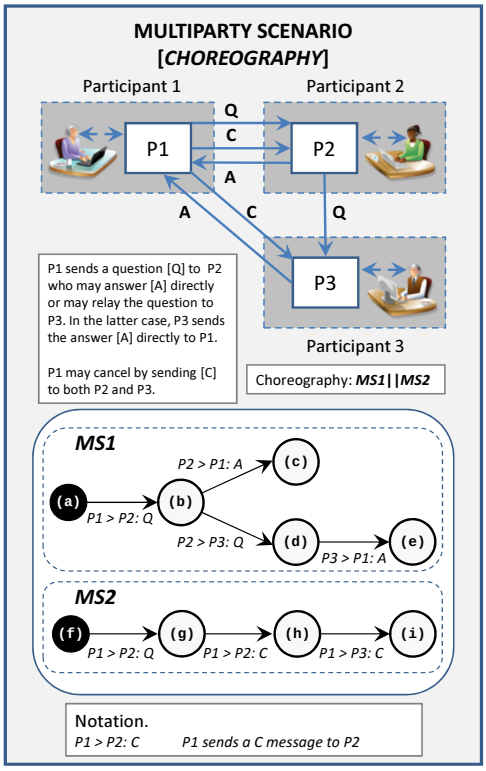

Figure 5: Multiparty Choreography

We assume that the collaboration is designed using a *global model* or *choreography* to describe the possible sequencing of interactions between the participants. The choreography for the example is shown in the lower part of Figure 5 and shows the possible orderings of sends of all the messages concerned with handling $P1$'s question. Note that the choreography does not define the ordering of message *receives*, which is constrained only by the fact that the receive of a given message cannot happen before the send and the receiver must be in a state in which reception of the message is possible. The choreography is represented as a composition of two machines, $MS1 \parallel MS2$. The use of composition allows the possibility of "message cross over": both participants sending simultaneously. For instance it is possible that $P1$ sends *Cancel* to $P2$ at the same time that $P2$ sends its *Answer* to $P1$.

Choreography theory supports analysis techniques to verify that collaboration based on the choreography are bound to be "successful" in the sense that whenever a message is sent, the recipient is guaranteed to be able to receive and process it eventually. So a message is never stuck in the network because the recipient never reaches a state that allows receipt. A choreography that can be verified to succeed this way is said to be *realizable*. A full discussion of choreography realizability is a significant area of research in itself and beyond the scope of this paper; however it is well explored by a number of authors, for instance: Fu et al. [22], Honda et al. [12], Lanese et al. [10] and McNeile [1]. The approach described in any of these papers could be used to construct the can-models of the participants of a distributed workflow, but we will use that described by McNeile. We assume that the number of participants in a workflow is fixed and finite,

and that every message send is between a specified sender and a specified receiver. More complex patterns of communication can be modeled by composition: "broadcasting" (where a process sends to a number of receivers) by a composition of multiple sends to specific receivers, and "racing" (where a process acts on the first receipt from a number of possible senders) by a composition of multiple receives from specific senders. See [1] for examples of both of these.

In the arguments we make concerning guaranteed completion of distributed workflow, we use two properties of choreographed collaboration, both of which are shown to be true of a realizable choreography in [1]:

- When one participant sends a message to another, the receiver is bound to be able to receive it with a delay that is determined only by platform latency. In other words, reception of a message that has been sent is never blocked because the receiver is waiting for another message that has not been sent.
- If the network is empty (as would be the case if participants were to refrain from sending until all "in flight" messages had been received), then all participants are synchronized on the same state of the choreography.

In the remainder of this section we discuss how to establish GC in the example in Figure 5. We consider this in four stages:

- Representing communication.
- The distributed can-model.
- The distributed want-model.
- Analysis of distributed workflow.
- Proof of Completion.

5.2 Representing Communication

In the context of distributed workflow the alphabet of actions of the can-model of participants now contains *communication actions* (both send and receive) used by the participant, as well its *local actions*. In Figure 5 *local actions*, which are interactions between the software of a participant and a local user, are shown as dotted arrows; and *communication actions* which represent communications over the network, shown as solid arrows.

Local actions are represented as in the earlier sections. Communication actions are represented as send actions, which are prefixed with "!" and receive actions which are prefixed with "?". This notion is borrowed from process algebras such as Hoare's CSP [5] and Milner's CCS [17]. However it is important to understand that, as we are assuming *asynchronous* communication, send and receive actions on a given message instance do not happen together: a message takes finite time to transit from sender to receiver through the network so the receive happens some time later than the send.

For the purposes of GC analysis in distributed workflow, we shall allow a participant's want-model to apply to any kind of action at that participant. The semantics of w-transitions relating to communication actions is as follows:

- A w-transition involving a *send action*, $!x$, will always happen immediately as we assume that the network is non-blocking. So when a participant wants (per its want-model) and is able (per its can-model) to send a message, it does so straight away.
- A w-transition involving a *receive action*, $?x$, must be shown to be resolved in finite time. If not, the distributed workflow is not guaranteed to complete.

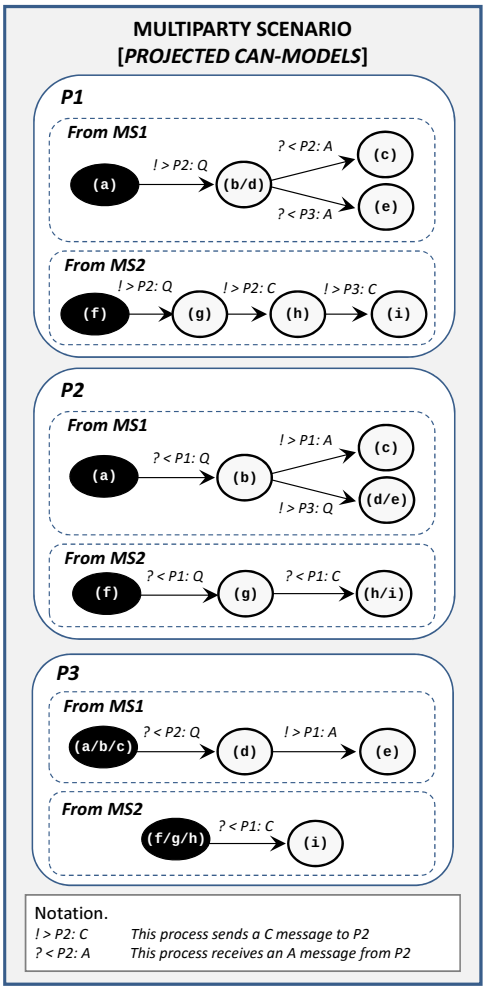

Figure 6: Projected Can-Models

We call a w-transition involving a receive action a *dependency*. A dependency is resolved when either the wanted message is received or, because of some other change of state in the model, the receive action ceases to be wanted. For instance, in the choreography shown in Figure 5, a dependency in $P1$ is established when a question is sent to $P2$. This would be represented as wanted receives for an answer, from either $P2$ or $P3$, in the model for $P1$. It is resolved once the choreography reaches a state that satisfies `(c or e or i)` as then the question has been answered (so the wanted receive has occurred) or canceled (so an answer is no longer wanted).

Proving that a distributed workflow will complete requires proving that dependencies will be resolved in finite time. Section 6.2 shows how this is done.

5.3 Distributed Can-Model

Each participant in a distributed workflow requires its own, local, can-model; so in Figure 5 each of {$P1$, $P2$, $P3$} will need a can-model which describes the possible sequence of actions of all types (local and communication) in which it may engage. As is the normal approach in designing choreographed collaborations, we will create individual can-models for the participants by extracting them from the choreography, a technique called *end-point projection*. This is a mechanical process that involves filtering the choreography to those transitions that each participant can know about, as we now describe.

A participant will only know about those exchanges in which it participates as sender or receiver. Thus, for instance, $P3$ cannot know whether or not $P1$ has sent a question or a cancel to $P2$ and so cannot tell the difference between states `(f)`, `(g)` and `(h)` of the choreography. Consequently, these three are combined in $P3$'s projection into a single state, shown as `(f/g/h)` in Figure 6. The set of states in the projection for a given participant represents that participant's view of the possible states of the choreography. The exchanges described in the choreography are projected into send and receive communication actions. For example the exchange $P1 > P2 : Q$ in the choreography becomes a send action $! > P2 : Q$ in the projection to $P1$.

The projections obtained by this technique are not complete definitions of the participants' behavior: they represent *behavioral contracts*, or *abstract processes*, to which the participants must conform in order to collaborate successfully. The can-models projected from the choreography thus provide that part of the definition of the behavior of participant that concerns its collaboration. These need to be combined with a local can-model that describes its local actions. Figure 8 shows the complete can-model for participant $P2$. It comprises three composed machines: the first two being those projected from the choreography (as shown in Figure 5) and the third showing the local can-model. The last of these shows that when it gets a question from $P1$, $P2$ asks a local user for the answer. If the local user is able to supply the answer (the action *Supply Answer*) this answer is relayed back to $P1$, but if the local user says *Don't Know* then $P2$ relays the question to $P3$.

5.4 Distributed Want-Model

Each participant in a distributed workflow also has its own want-model. The actions that can be "wanted" by the want model include all types (local and communication) in which it may engage. In particular, we allow both send ($!x$) and receive ($?x$) communication actions to be wanted.

The technique to ensure that dependencies are resolved is to engineer want-models for the participants to "motivate the choreography". So we arrange that, while the dependency in $P1$ is *unresolved*, the want-models drive the choreography forward to *resolve* it. This is done making the *send actions* of the choreography wanted, appropriately for each state of each participant. Figure 7 shows this. For example, $P2$ has two send actions in its projection in Figure 6 and we make these wanted in $P2$'s want-model in Figure 7 when the state indicates that they are outstanding. We will *prove* that we have been successful in motivating the choreography by showing, in Section 6.1, that the choreography has GC.

In a similar manner to the can-model approach, the want-models associated with the choreography need to be combined with local want-model that describe the motivation for the local (non communication) actions. These are straightforward and we have not shown them.

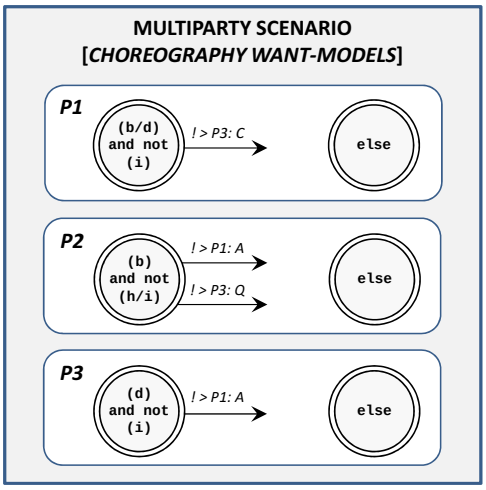

Figure 7: Choreography Want-Models

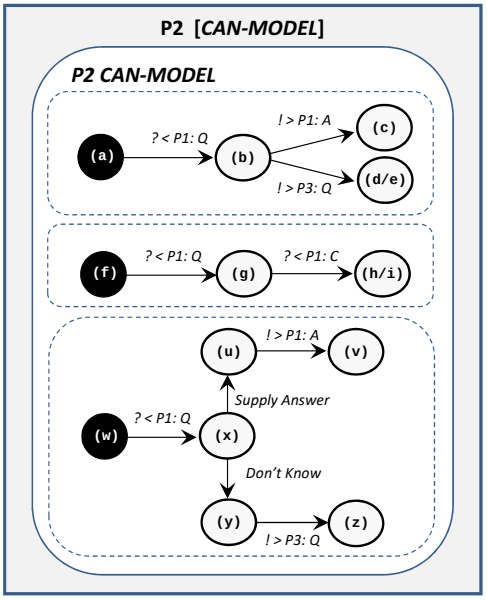

Figure 8: P2 Can-Model

6. DISTRIBUTED COMPLETION ANALYSIS

We establish GC in a distributed workflow by using a combination of:

- GC analysis of each participant to show that, provided dependencies are resolved, each participant will complete; and
- GC analysis of the choreography used between the participants to show that dependencies will be resolved.

This means that, in the example above, a total of four separate GC analyses are required: one for each the three partic-

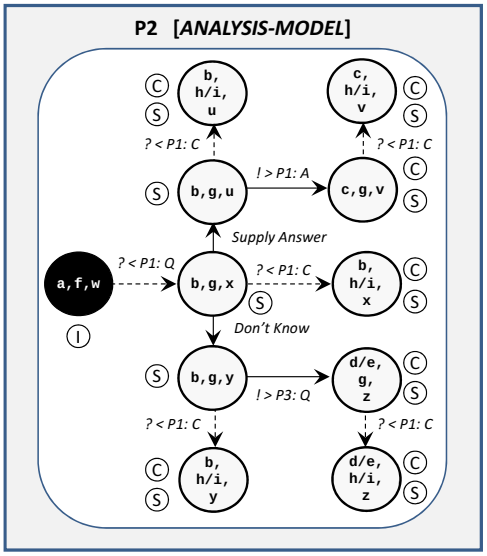

Figure 9: P2 Analysis-Model

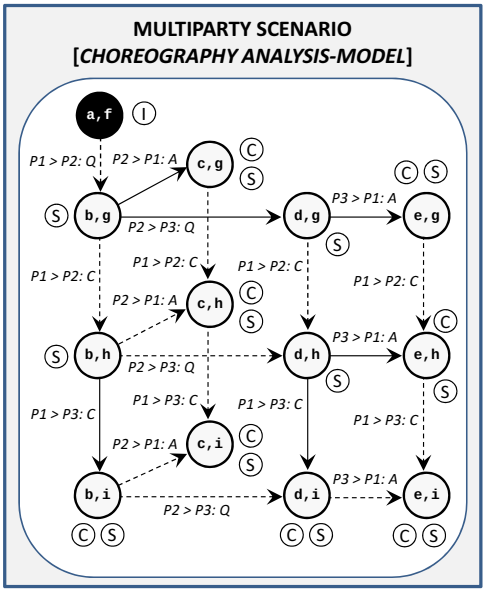

Figure 10: Choreography Analysis-Model

ipants and one for the choreography. The GC analysis of the participants must take into account both their local want-models and the want-models imposed on them by the need to ensure completion of the choreography. To show how this works, we look at two of the four GC analyses required in the example: that for $P2$ and that for the choreography.

6.1 Analysis for *P2* and the Choreography

The GC analysis for $P2$ is shown in Figure 9. This uses the want-model from the choreography, shown in Figure 7, and also a local want-model (not shown) that makes *Supply Answer* and *Don't Know* wanted if the can-model is in state `(x)`. We then check the following:

1. That $P2$ has GC. This is clear from Figure 9.

2. That every Completion State, those marked Ⓒ, is also a business completion. We note that the Completion States all satisfy the predicate `(v or z or h/i)` and it is clear from $P2$'s can-model that these are business completions as $P2$ has either answered or relayed the question, or received a cancel from $P1$.

Now we analyze the *choreography* in Figure 5. We do this exactly as described earlier for a non-distributed workflow:

- Combine $MS1$ and $MS2$ into a single topological representation of the choreography. This is the analysis-model for the choreography.

- Use the want-models in Figure 7 to classify each transition in the combined representation of the choreography as either a c-transition or a w-transition. A w-transition is one where the sender wants to send in that state of the choreography.

- Classify the states of the choreography according to Section 4.2. The Completion States are those in which no further send is wanted by any participant.

This is done in Figure 10. We then check the following:

1. That the choreography obeys the well-formedness rules specified in Section 4.3. In particular, for Condition 4, we ensure that every Completion State of the choreography is one in which all dependencies are satisfied. The only dependency in this workflow is that in $P1$ (on receiving an answer to its question) and, as noted in Section 5.2, this is resolved in any state that satisfies `(c or e or i)`. We note that every Completion State in Figure 10 satisfies this predicate.

2. That the choreography has GC. This is clear from Figure 10.

6.2 Proof of Completion

Once we have established GC in all participants and the choreography we can argue that the distributed workflow is bound to complete in finite time.

Now suppose that we have a distributed workflow and we have established GC in the participants and in the choreography. First we assume that:

- The network does not lose messages, so any message sent will be available for receipt within finite time.

- For the purposes of reasoning about states of the collaboration we can take it that there are no unconsumed messages in the network. This is because, at any point of execution, we can wait a finite time for unconsumed messages to be received, which is always possible in a choreographed collaboration (and is the basis of the realizability proof in [1]).

We assume that not all participants have completed (reached a Completion State) but that nevertheless, the workflow cannot proceed in finite time. We show that this leads to a contradiction, as follows.

1. The only way the workflow can be unable to proceed is if one or more participants is in a Safe State from which the only exit is by an unresolved wanted receive,

as any other w-transition in a participant is bound to happen in finite time. For at least one such receive in one such participant, the send that resolves the receive must be wanted in the sender. If this were not the case, and no send were wanted, there would be no w-transition from the current state of the choreography. The choreography would then be in an uncompleted state with no exit w-transition, which is not possible as the choreography has GC.

2. Now consider the sender in which this send is wanted. By assumption, this participant has GC. This participant cannot be in Completion State, as no outstanding wants are allowed at completion. Nor can it be blocked in a Safe State from which the only exit is an unresolved receive, as in this case the send would not be possible in the choreography, which we know it to be as it is a w-transition in the choreography. So either the send itself can take place or some other w-transition, based on a local action, can take place. Either way, the distributed workflow is able to proceed, in violation of the original assumption.

This means that there must be a w-transition that will go ahead and advance the workflow, in contradiction to the assumption that the workflow cannot proceed. It is therefore impossible for the workflow not to proceed, and so all participants must complete. Because by definition of Completion State to send actions can then be wanted once all participants have completed, the choreography must complete too.

On the assumption that:

- All local actions are subject to an OC that bounds the time in which it can be outstanding.
- All w-transitions for sends happen immediately as the network is non-blocking.
- The network has finite (bounded) latency so that a message once sent is available to receive after finite time.

Then a distributed work-flow that has GC in each of its participants and in its choreography will have GC overall.

7. RELATED WORK

The introduction of the concept of motivation, and construction of a notion of progress based around motivation, is a central theme of this paper. As far as we know our work is the first to explore this idea. The focus of this section is therefore to explain the differences between our work and the other main approaches to the description and analysis of workflow in the published literature.

7.1 Reachability and Soundness

The most common notion used in the analysis of workflow properties is that of *reachability* and the related one of *soundness*, in particular as summarized by van der Aalst et al. [21]. This kind of analysis determines whether or not a workflow *can* reach a specified terminal state or states, and thus whether it *can* complete. No distinction between solicited and non-solicited actions is needed in the context of reachability analysis, as it is performed on the basis of the can-model alone. This is a weaker result than the one to which we aspire, as we aim ensure that completion is *certain*, not just *possible*.

7.2 Liveness and Fairness

Concepts of *liveness* and *fairness* are stronger than reachability, being concerned with whether a system is *bound* to reach a desired state; and such questions clearly bear resemblance to the question of whether a workflow is bound to complete. Liveness and fairness have been studied widely and this work has spawned a variety of definitions and analytical approaches. The main focus of the work has been the study of models that are *non-deterministic* as the result of concurrency, as discussed by Kindler [7]. Here the challenge is to establish sufficient constraints on the non-deterministic choices made in the system to ensure liveness. As papers in this area (such as those by Owicki and Lamport [18] and by Giannakopoulou et al. [6]) explain, this requires that the non-deterministic choices obey *fairness constraints* that place conditions on sequences of choices without determining individual choices. Because, the non-determinacy in our workflow models is of a very limited kind, this form of fairness does not apply to our work:

- The first form of non-determinism in our models is that associated with which action, of all those possible, takes place next. (This is sometimes called *external non-determinism.*) However, we are *not* concerned to determine constraints on this non-determinism to achieve liveness (completion). Rather, we are concerned to show that under *all* possible choices of next action (provided OCs are obeyed) the workflow will complete.
- The second form of potential non-determinism in our models is from race-conditions in distributed workflow, associated with network latency. We eliminate this non-determinism by using choreography techniques as the basis for the design. Using a choreography controls the concurrency sufficiently to allow liveness (guaranteed completion) to be established without the need for any assumption about how the network operates (beyond requiring that messages do get lost) as we show in Section 6.2.

Note that there is a more limited meaning of the term *fair* that is associated not with non-determinism but with loops. This is the sense in which, for instance, Kindler uses the term in his discussion of workflow [8]. This, different, notion of fairness only concerns that cyclic behavior is limited; and this issue is relevant to our work as discussed in [2]. Our treatment there is similar to that by Kindler.

7.3 Time in Workflow Models

Our discussion of timing, as given in Section 4.5, is very basic. Other authors have considered time and temporal constraints in workflow using more sophisticated approaches, in particular:

- Modeling workflow using variants of standard modeling formalisms in which the states of the model encode time as well as data. Execution traces then explicitly represent the progress of time, as well as the evolution of the data and state of the process, allowing standard model proving techniques to explore temporal properties of the workflow. This approach is used, for example, by Kazhamiakin et al. [16] who introduce a timed version of Labeled Transition Systems; and by van der

Aalst [20] who embeds a representation of time into Petri Net modeling.

- Layering a *temporal constraint language* over the top of a process model to describe and reason about the ability of a process to meet timing requirements. This normally involves arguing about whether constraints in the small (at the level of activities) are compatible with constraints in the large (end-to-end). This approach is explored, for instance, by Bettini et. al [4] and by Marjanovic and Orlowska [15].

Both such approaches could be used with the modeling approach described in this paper to provide tools that allow a more sophisticated treatment of time. This would require extension or adaption of the techniques to handle the difference between c-transitions and w-transitions, but there seems to us no reason in principle why this should not be done.

7.4 Modal Specification

There is apparent similarity between our scheme and formalisms that allow a modal dimension to the specification of behavior: two examples are *Modal Transition Systems* (MTS) and *Live Sequence Charts* (LSC). However the semantics of both MTS and LCS differ significantly, but in different ways, from the semantics of our motivation modeling.

In MTS [13] transitions are classed as *required*, *possible* or *not possible* in a manner that appears similar to the scheme using can- and want-models described here. However the distinctions in MTS refer to the degree of *certainty in a specification*, rather an expression of motivation in the final system. As D'Ippolito et al. say when describing MTS: *In MTS, each transition can be either required or maybe. The latter means that it is not yet certain if the interaction modeled by the transition is required or prohibited in the final system.* [14]. The use of *maybe* in MTS therefore reflects looseness or incompleteness in a specification which is expected to be tightened by refinement. This is clearly quite different from the semantics we describe.

The LCS [19] formalism extends Message Sequence Charts in various ways, including the ability to specify whether behavior beyond a location in the chart is mandatory or "provisional". If a location on a LSC is designated *hot*, behavior depicted beyond that point is bound to happen; whereas if a location is designated *cold* behavior is possible but not guaranteed. The guarantee of occurrence of mandatory behavior is semantically similar to the fulfilment of a post-condition, where arrival at the hot location forms the pre-condition; so failure of mandatory behavior to take place represents a failure of the system to meet its specification. This contrasts with the semantics of a w-transition which is only bound to happen in so far as the external agent obeys her OC. Non occurrence could be because some other transition happens first and does not impugn the model in any way.

Perhaps the key point is that motivation, as embodied in the want-model, concerns a relationship between the software system and the agents outside who interact with it. Model specification formalisms (such as MTS and LCS) are not - they are only concerned with the specification of the software itself.

7.5 Workflow Exceptions

An area that does bear some similarity to our work is that on "workflow exceptions", such as that of Eder et al.[11] and Casati et al.[9]. This work discusses the idea of an *expected exception* - a situation that is known to be possible and allowed for in the design of the workflow, but which is regarded as a departure from the "normal flow" of the workflow process. Eder et al. characterize expected exceptions as *something which does not represent the "normal" case but still may arise frequently and therefore special mechanisms are available to handle such special cases.* No formal means is given to distinguish *expected* from *unexpected* actions, nor is the characterization based on any concept of motivation (the way the workflow acts to solicit actions using a "to-do list" or equivalent mechanism). The concerns of the authors are more related to the expressiveness of workflow definition languages rather than to provide a basis for the analysis of formal properties.

7.6 Collaboration and Distributed Workflow

The current trend is to handle distributed processing using a choreography. As we point out in Section 5.3, the process of projecting end-point behaviors from a choreography provides a can-model for each participant such that they can participate successfully in the collaboration, and this is discussed in the current literature (see the references in Section 5.1). However, because it is only providing a can-model, this only ensures the *possibility* of completion. To *guarantee* completion requires a concept of motivation, whereby actions are solicited and are then bound to happen. No other authors have yet explored this area.

Some other work in choreography could provide a suitable basis for modeling distributed workflow. For instance, work by Qiu et al. [23] considers distributed systems using choreography expressed in terms of both communication and local actions which could provide a basis for completion analysis. In addition, their work includes consideration of non-determinism and it would be interesting to see if our analysis could be extended to work with non-deterministic collaborative behaviors. This is a possible topic for future research.

8. CONCLUSION

The increasing focus on business process management, service based collaborations and outsourcing means that the ability to ensure that workflows will complete and thus support Service Level Agreements is increasingly important. We have shown how modelling *motivation* in workflow, as an active agent in ensuring progress and completion, can be used in conjunction with *choreography* techniques to allow completion and timing analysis in distributed multiparty workflows operating over a network.

The ideas presented here can contribute to the development of more powerful techniques for the design and verification of e-commerce applications and cross-enterprise workflow to ensure that they will always progress to completion. Such techniques are particularly needed where service level agreements require guaranteed response and non-compliance has financial penalty implications.

References

[1] A. McNeile. Protocol Contracts with Application to Choreographed Multiparty Collaborations. *Service Ori-*

ented Computing and Applications, 4(2):109–136, June 2010.

[2] A. McNeile and E. Roubtsova. Motivation and Guaranteed Completion in Workflow. In B. Shishkov, editor, *Business Modeling and Software Design*, volume 142 of *Lecture Notes in Business Information Processing*, pages 16–42. Springer Berlin Heidelberg, 2013.

[3] A. McNeile and N. Simons. Protocol Modelling: A Modelling Approach that supports Reusable Behavioural Abstractions. *Journal of Software and System Modeling*, 5(1):91–107, 2006.

[4] C. Bettini, X. Sean Wang and S. Jajodia. Temporal Reasoning in Workflow Systems. *Distrib. Parallel Databases*, 11:269–306, May 2002.

[5] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[6] D. Giannakopoulou, J. Magee and J. Kramer. Checking Progress with Action Priority: Is it Fair? In O. Nierstrasz and M. Lemoine, editor, *Software Engineering Ů ESEC/FSE Š99*, volume 1687 of *Lecture Notes in Computer Science*, pages 511–527. Springer Berlin / Heidelberg, 1999.

[7] E. Kindler. Safety and Liveness Properties: A Survey. *EATCSBulletin*, 53(53):268–272, 1994.

[8] E. Kindler, A. Martens and W. Reisig. Inter-operability of Workflow Applications: Local Criteria for Global Soundness. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 235–253, London, UK, 2000. Springer-Verlag.

[9] F. Casati, S. Ceri, S. Paraboschi and G. Pozzi. Specification and implementation of exceptions in workflow management systems. *ACM Trans. Database Syst.*, 24:405–451, September 1999.

[10] I. Lanese, C. Guidi, F. Montesi and G. Zavattaro. Bridging the Gap between Interaction- and Process-Oriented Choreographies. In *Proceedings of SEFM'08, 6th IEEE International Conferences on Software Engineering and Formal Methods*, pages 323–332, Washington, DC, USA, 2008. IEEE Computer Society.

[11] J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proceedings of the 3rd international conference on Cooperative Information Systems (CoopIs'95)*, pages 87–98, 1995.

[12] K. Honda. N. Yoshida and M. Carbone. Multiparty Asynchronous Session Types. *SIGPLAN Not.*, 43(1):273–284, 2008.

[13] K. Larsen and B. Thomsen. A Modal Process Logic. In *LICS*, pages 203–210, 1988.

[14] N. D'Ippolito, D. Fishbein, H. Foster and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *eclipse '07: Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 6–10, New York, NY, USA, 2007. ACM.

[15] O. Marjanovic and M. Orlowska. On Modeling and Verification of Temporal Constraints in Production Workflows. *Knowl. Inf. Syst.*, 1(2):157–192, 1999.

[16] R. Kazhamiakin, P. Pandya and M. Pistore. Timed Modelling and Analysis in Web Service Compositions. In *International Conference on Availability, Reliability and Security*, pages 840–846, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[17] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[18] S. Owicki and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982.

[19] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Form. Methods Syst. Des.*, 19(1):45–80, 2001.

[20] W. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, pages 453–472, London, UK, 1993. Springer-Verlag.

[21] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve and M. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, pages 1–31, 2010.

[22] X. Fu, T. Bultan and J. Su. Conversation protocols: a Formalism for Specification and Verification of Reactive Electronic Services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.

[23] Z. Qiu, X. Zhao, C. Cai and H. Yang. Towards the Theoretical Foundation of Choreography. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 973–982, New York, NY, USA, 2007. ACM.