Um Cálculo para a Modelação e Análise de Conversações em Computação Orientada a Serviços

Hugo Torres Vieira
Resumo da Dissertação de Doutoramento

Junho, 2010

Sumário. A disseminação do uso da tecnologia de serviços Web tem motivado um crescente interesse no paradigma da computação orientada a serviços. Existem muitos desafios específicos a este paradigma que requerem novos conceitos, modelos especializados e técnicas adequadas. Este documento visa resumir o trabalho desenvolvido para endereçar alguns destes desafios, nomeadamente a nível de um modelo de especificação baseado numa noção inovadora de conversação e técnicas de verificação que permitem garantir propriedades cruciais dos sistemas: que todos os participantes seguem os protocolos prescritos — *fidelidade à conversação* — e ausência de pontos de bloqueio — *progresso na conversação*. Na nossa abordagem é possível modelar e analisar sistemas que envolvem várias conversações simultâneas entre múltiplas partes e com um número dinâmico e imprevisto de participantes, cenários que podemos encontrar em sistemas reais e que estão fora do alcance de trabalhos anteriores.

1 Introdução

A grande disseminação do uso da tecnologia de serviços Web suscitou um imenso esforço de investigação dedicado ao paradigma da computação orientada a serviços nos últimos anos [1]. Para melhorar a qualidade das aplicações desenvolvidas, nomeadamente para eliminar os erros em tempo de execução (tais como "crashes", bloqueios e mensagens fora de contexto), tornou-se necessário o desenvolvimento de standards, ferramentas e técnicas especializadas que contribuíssem para o desenvolvimento de tais sistemas. Existem muitos novos desafios que se colocam no estudo de aplicações orientadas a serviços que requerem novos conceitos, modelos especializados e técnicas de verificação adequadas. O trabalho apresentado neste documento insere-se neste esforço, endereçando, em particular, as problemáticas associadas à especificação e análise da comunicação numa aplicação orientada a serviços.

Um dos aspectos inovadores do trabalho aqui apresentado prende-se com a forma como é modelada a comunicação entre vários participantes. Uma das abordagens tradicionalmente propostas para modelar a comunicação em sistemas orientados a serviços é baseada na noção de *sessão*. Essencialmente, uma sessão caracteriza a interacção entre dois participantes, um cliente e um servidor, que trocam mensagens entre si de uma forma sequencial e dual. A noção de sessão é então particularmente adequada ao paradigma cliente/servidor, contudo trata-se de uma abordagem que fica aquém quando se trata de modelar a comunicação entre vários participantes, um cenário frequente em sistemas orientados a serviços reais. A abordagem aqui descrita inova de forma decisiva o estado da arte pois permite modelar a comunicação entre vários participantes de uma forma igualmente canónica, mas baseada numa inovadora noção de *conversação*: uma extensão simples da noção de sessão que permite que múltiplos participantes interajam no mesmo meio de

comunicação de uma forma disciplinada, através de mensagens etiquetadas.

As contribuições deste trabalho são as seguintes. Em primeiro lugar, propomos um modelo de especificação formal para computação orientada a serviços, o Cálculo das Conversações, baseado na nossa noção de conversação, que permite agrupar todas as comunicações entre as várias partes que dizem respeito à mesma tarefa de serviço. Em segundo lugar, propomos técnicas formais, nomeadamente o sistema de tipos de conversações e o sistema de prova de progresso que, baseando-se na noção abstracta de conversação, permitem garantir boas propriedades dos sistemas em tempo estático (i.e., antes que as aplicações sejam executadas) tais como "os protocolos previstos vão ser seguidos pelos intervenientes" — *fidelidade à conversaç*ão — e "o sistema nunca irá entrar num estado bloqueado" — *progresso na conversação*. Mostramos, de forma substancial, que na nossa abordagem é desde já possível modelar e tipificar aplicações orientadas a serviços bastante sofisticadas, a um nível de abstracção relativamente elevado. Exemplos de tais sistemas incluem cenários que envolvem várias conversações simultâneas entre múltiplos participantes, com concorrência e acesso a recursos locais, e conversações com um número dinâmico e imprevisto de participantes, configurações que podemos encontrar em sistemas reais e que estão fora do alcance de trabalhos anteriores. O objectivo último deste trabalho é vir a influenciar o desenvolvimento de ferramentas e linguagens que permitam ao programador evitar erros de concepção.

No resto deste documento apresentamos as principais ideias que estão na base da nossa proposta. Em primeiro lugar apresentamos o contexto em que o trabalho se insere, através de uma breve descrição do estado da arte dos standards da indústria para tecnologia orientada a serviços, para sugerir ao leitor algumas da problemáticas que motivam o nosso desenvolvimento. Depois, apresentamos em linhas gerais a linha de investigação na qual o nosso trabalho se insere, e assinalamos alguns dos problemas em aberto, donde que não estão ao alcance de abordagens relacionadas, de forma a identificar os pontos onde se inserem as nossas contribuições. Em segundo lugar, introduzimos os nossos principais avanços técnicos: começamos por apresentar de que forma é que a nossa abordagem difere de abordagens relacionadas de forma a ir de encontro aos problemas em aberto mencionados. Depois, descrevemos as principais características da computação orientada a serviços que estão na base do nosso desenvolvimento. Segue-se a apresentação da linguagem que propomos para modelar computação orientada a serviços — o Cálculo das Conversações — através da descrição pormenorizada das várias primitivas, da apresentação da semântica da linguagem e da análise detalhada de um exemplo. Finalmente introduzimos as principais ideias que estão a base das nossas técnicas de análise — o sistema de tipos de conversações e o sistema de prova de progresso — e enunciamos os principais resultados. Por fim, listamos as contribuições do trabalho e indicamos algumas direcções para trabalho futuro.

2 Contexto do Trabalho

Nesta secção identificamos o contexto onde o trabalho se insere, começando por uma perspectiva mais tecnológica, de forma a assinalar algumas das problemáticas associadas ao paradigma da computação orientada a serviços que motivam o nosso trabalho. Depois apresentamos a linha de investigação onde o trabalho se insere e descrevemos alguns problemas em aberto de forma a identificar os pontos onde se inserem as nossas contribuições.

2.1 Desenvolvimento de Software Orientado a Serviços

O principal objectivo do paradigma da computação orientada a serviços é suportar o desenvolvimento de sistemas complexos a partir de aplicações mais simples que estão distribuídas na rede. A motivação passa por argumentos recorrentes no desenvolvimento de software: (re)utilizar funcionalidades que já estão disponíveis para desenvolver aplicações, em vez de programar tudo de raiz.

É também um dado adquirido que implementar um sistema complexo através da separação das funcionalidades em módulos especializados contribui para a fiabilidade dos sistemas, pois estes módulos podem ser testados e analisados individualmente. Mais, uma tal modularização facilita a substituição de partes do sistema, bastando que os interfaces externos se mantenham, permitindo uma mais fácil manutenção e evolução dos sistemas desenvolvidos. Se juntarmos a este conjunto de argumentos mais "clássicos" o facto que os serviços executam remotamente e portanto usam recursos remotos para executar as suas tarefas, então obtemos um paradigma de desenvolvimento de software muito aliciante.

Uma aplicação orientada a serviços é construída a partir da colaboração de vários parceiros, cada um contribuindo com uma parte da funcionalidade global. Tais colaborações são estabelecidas tipicamente de forma dinâmica e sem controlo centralizado. Standards da indústria propostos para a tecnologia de serviços Web tais como UDDI [4] ou WS-Discovery [3] suportam a descoberta dinâmica de serviços, enquanto SOAP [18] define um protocolo que suporta a troca de informação de forma descentralizada entre serviços Web. Uma preocupação central no desenvolvimento de sistemas orientados a serviços é então o desenho de protocolos de comunicação que permitam a colaboração, de forma descentralizada e dinâmica, entre vários parceiros.

O standard "Web Services Choreography Description Language" (WS-CDL) [22], ou linguagem de descrição de coreografias para serviços Web, é uma das propostas de linguagens que permitem a especificação de interacção entre serviços Web. O termo coreografia reflecte a ideia de um cenário descentralizado e dinâmico: os parceiros interagem como se fossem bailarinos num palco que, graças a uma bem definida coreografia, conseguem levar a cabo uma exibição sem colisões ou encontrões. Uma tal linguagem permite estruturar os protocolos de interacção e raciocinar sobre a sua correcção. A nível de linguagens de programação que suportem a especificação de serviços Web encontramos propostas tais como "Web Services Business Process Execution Language" (WS-BPEL) [2], ou linguagem executável de processos de negócio para serviços Web. Do ponto de vista de um programador seria interessante poder, dado um programa WS-BPEL, verificar se o programa cumpre os protocolos previstos numa coreografia WS-CDL, de forma a transportar as propriedades garantidas a nível da análise dos protocolos para a implementação. Problemas desta natureza estão na base de abordagens como a de Carbone et al. [11], que apresenta uma metodologia para projectar especificações globais de interacção (semelhantes a especificações WS-CDL) em especificações de processos locais, de forma a garantir, por construção, que os processos locais cumprem a coreografia definida globalmente.

Uma preocupação central no desenvolvimento de sistemas orientados a serviços pode então ser expressa pela seguinte pergunta: "será que este sistema orientado a serviços está implementado através de colaborações que seguem protocolos de comunicação bem definidos?". Em geral, esta pergunta levanta problemas difíceis de análise visto que o alvo do estudo são sistemas extremamente dinâmicos e sem controlo central, tendo em conta que a análise tem de, em qualquer modo, contabilizar todos os comportamentos que podem surgir durante a execução das aplicações, apenas olhando para o código fonte das mesmas. A tarefa fica ainda mais complicada, para não dizer impossível, quando a semântica das linguagens de programação usadas está definida de forma descritiva e informal. Quando assim é, no limite, devido a interpretações diferentes por parte dos programadores dos compiladores das linguagens, podem obter-se comportamentos diferentes a partir do mesmo código fonte, o que impossibilita a sua correcta análise.

Esta discussão informal serve para introduzir a abordagem usada no trabalho aqui apresentado:

> A nossa abordagem consiste no desenvolvimento de modelos equipados com uma semântica definida matematicamente e técnicas formais de análise sobre tais modelos, que suportem mecanismos que forneçam respostas de forma automática sobre propriedades cruciais dos sistemas desenvolvidos, em tempo estático, ou seja durante a compilação dos programas.

Tipicamente, estes modelos e técnicas não são directamente usáveis na prática, pois necessariamente têm de abstrair de muitos detalhes para que as teorias sejam tratáveis e apresentáveis. Contudo, o seu desenvolvimento visa formar a base teórica de linguagens de programação e tecnologias de desenvolvimento de software que possam ser usados na prática, donde, para que tal possa ser possível, é importante apontar sempre a teorias gerais, simples e legíveis.

Os modelos de especificação dos programas devem então ser equipados com uma interpretação matemática rigorosa para que possam ser analisados. Por sua vez, técnicas de análise tais como os sistemas de tipos, que podem ser vistos como lógicas especializadas na caracterização do comportamento dos programas, podem então ser usados para extrair a informação necessária que permita provar as propriedades de interesse, nomeadamente a ausência de erros. Então, se por um lado os tipos são determinantes para garantir a ausência de erros, estes podem também ser usados para capturar informação dos programas que permite raciocinar sobre os mesmos. Este tipo de abordagens têm vindo a influenciar e melhorar linguagens "mainstream" tais como java ou ML.

2.2 Modelos de Especificação de Comunicação em Sistemas Concorrentes

O trabalho pioneiro de Milner nos anos 80 pode ser visto como dos mais influentes nas origens da área de investigação de modelos de comunicação (tendo sido reconhecido com a atribuição do Prémio Turing em 1991). O "Calculus of Communicating Systems" (CCS), ou cálculo de sistemas comunicantes, proposto por Milner em 1980 [27], apresentava desde logo os elementos básicos de um modelo de interacção de uma forma minimal e elegante e, mais importante ainda, propunha uma interpretação matemática para estes ingredientes básicos. Apesar de não abarcar uma noção explícita de distribuição, pois não contempla localizações ou "sites", apresenta um modelo geral para computação concorrente, onde a distribuição física pode ser vista como um aspecto ortogonal. Um dos ingredientes básicos para especificar interacção concorrente é a composição paralela de processos, denotada em CCS por $ProcessoA \mid ProcessoB$, que representa que o $ProcessoA$ e o $ProcessoB$ estão a executar em paralelo ou simultaneamente. Outro ingrediente é um mecanismo de comunicação que permita que processos paralelos sincronizem, nomeadamente capacidades de acção onde exista uma noção de dualidade, como por exemplo envio e recepção. Em CCS especificamos por $a.ProcessoA$ um processo que está pronto para executar a acção a, após a qual irá proceder de acordo com o que é especificado por $ProcessoA$. Considerando que $\overline{a}$ é a acção dual de a então temos que:

$$a.ProcessoA \mid \overline{a}.ProcessoB$$

especifica a composição paralela de dois processos que estão prontos para executar acções duais. Então, eles podem sincronizar e evoluir para a configuração $ProcessoA \mid ProcessoB$, um passo de evolução que muitas vezes é descrito com a ajuda do símbolo $\rightarrow$ ($ProcessoA \rightarrow ProcessoB$ significa que $ProcessoA$ evolui, num passo de execução, para $ProcessoB$), donde a evolução do processo acima é capturada por:

$$a.ProcessoA \mid \overline{a}.ProcessoB \rightarrow ProcessoA \mid ProcessoB$$

Outra primitiva permite especificar comportamento alternativo, o operador de escolha. A especificação $a.ProcessoA + outro\_a.ProcessoB$ denota um processo que pode executar a acção a e nesse caso prosseguir como $ProcessoA$, ou então executar a acção $outro\_a$ e nesse caso proceder como $ProcessoB$. Por exemplo, se colocado num contexto onde existe outro processo que pode executar a acção dual de a obtemos o seguinte comportamento:

$$\overline{a}.ProcessoC \mid a.ProcessoA + outro\_a.ProcessoB \rightarrow ProcessoC \mid ProcessoA$$

As acções em CCS são representadas por nomes e as suas acções duais pelos seus "co-nomes". É possível modelar em CCS que determinados nomes são locais a uma parte do processo. Por

exemplo, no processo seguinte:

$$ProcessoC \mid (\boldsymbol{\nu} a)(a.ProcessoA \mid \overline{a}.ProcessoB)$$

especificamos por $(\boldsymbol{\nu} a)$ que o nome a é local à parte do sistema $a.ProcessoA \mid \overline{a}.ProcessoB$ estando "escondido" do $ProcessoC$. Desta forma, é possível modelar interacções locais a partes do sistema. Na realidade, esta é a única forma de modelar distribuição em CCS, pois permite induzir uma noção de configuração espacial: se um nome é conhecido apenas por uma parte do sistema e não pode ser falsificado por outras partes, então podemos identificar o subsistema correspondente à parte do sistema que conhece o nome e outro subsistema correspondente à parte que não conhece o nome. Tais configurações espaciais podem têm necessariamente de ser directamente mapeadas em distribuição física, sendo mais gerais e permitindo capturar noções de distribuição lógica.

O modelo CCS não permite capturar sistemas dinâmicos, pois as suas configurações espaciais são definidas através destes nomes escondidos de uma forma estática. Este facto motivou o desenvolvimento do Cálculo-π [29], que introduziu a capacidade de comunicar nomes nas mensagens. Desta forma, o Cálculo-π permite que as configurações espaciais sejam dinamicamente definidas pois os nomes que as definem podem ser comunicados entre os vários intervenientes do sistema. Por exemplo, na configuração seguinte:

$$\begin{array}{l} vamosFalar(x).\overline{x}(\textit{"olá"}) \\ \mid (\boldsymbol{\nu}\, conversaPrivada)(\overline{vamosFalar}(conversaPrivada).conversaPrivada(msg)) \end{array}$$

que representa a composição paralela de dois processos, onde o processo em cima está à espera de receber um nome através da comunicação em $vamosFalar$ (identificado pela variável x), procedendo após a recepção ao envio de uma mensagem de texto através do canal de comunicação recebido. O processo em baixo, por sua vez, está pronto para enviar um nome privado ($conversaPrivada$) na comunicação em $vamosFalar$, ficando depois à espera de receber uma mensagem neste canal privado. Através de uma sincronização em $vamosFalar$ o nome $conversaPrivada$ é passado entre os processos, permitindo ao processo em cima conhecer de forma dinâmica o nome escondido:

$$\begin{array}{l} vamosFalar(x).\overline{x}(\textit{"olá"}) \\ \mid (\boldsymbol{\nu}\, conversaPrivada)(\overline{vamosFalar}(conversaPrivada).conversaPrivada(msg)) \\ \rightarrow \\ (\boldsymbol{\nu}\, conversaPrivada)(\overline{conversaPrivada}(\textit{"olá"}) \mid conversaPrivada(msg)) \end{array}$$

O escopo do nome escondido (ou restrição de nomes na terminologia usada no Cálculo-π) cresce como consequência da comunicação — tipicamente descrito como extrusão de escopo. Esta capacidade de modelar sistemas com reconfigurações dinâmicas através da mobilidade dos nomes elevou o estatuto do Cálculo-π para o modelo de referência para especificar sistemas concorrentes e dinâmicos, tendo motivado inúmeros trabalhos de investigação na especificação e análise de sistemas concorrentes. O Cálculo-π também influenciou o desenvolvimento de várias linguagens de programação protótipo (por exemplo [5, 17]).

Entre as técnicas de análise mais usadas em linguagens de programação em geral identificamos os sistemas de tipos. Os tipos suportam mecanismos que evitam erros de execução, sendo por vezes também úteis para extrair informação sobre os programas que ajuda a raciocinar sobre o comportamento em tempo de execução dos mesmos. Para o Cálculo-π em particular existem muitas propostas de sistemas de tipos com variadas finalidades. O primeiro sistema de tipos para o Cálculo-π foi o sistema de géneros proposto por Milner [28], que endereçava o problema de garantir que as comunicações são definidas de forma homogénea no que toca à aridade dos tuplos comunicados. Posteriormente, outras disciplinas de tipos foram propostas para endereçar o uso

consistente dos canais para recepção e envio (por Pierce et al. [30]), para garantir condições de linearidade no uso dos nomes (por Kobayashi et al. [24]), entre outras. Seguiram-se sistemas de tipos mais sofisticados para o Cálculo-π ao longo dos anos. Exemplos incluem propostas onde os tipos que caracterizam o comportamento dos sistemas formam um modelo de processos completo [12], propostas de modelos genéricos para sistemas de tipos para o Cálculo-π [21] e propostas com finalidades mais finas tais como garantir ausência de estados bloqueados (ausência de "deadlocks") [23], ou que visam garantir o correcto uso dos recursos do sistema [25].

Uma referência de relevo para sistemas de tipos que visam estruturar as interacções de sistemas concorrentes centrados em comunicação especificados em Cálculo-π é a abordagem dos tipos de sessão, propostos por Honda et al. [19]. Os tipos de sessão caracterizam a interacção entre duas entidades, tipicamente um cliente e um servidor, que interagem de forma sequencial e dual num meio de comunicação dedicado. Os tipos de sessão influenciaram o desenvolvimento de várias propostas em variados campos da informática, tais como programação orientada a objectos [15] e programação funcional [31]. Também influenciaram algumas abordagens de mais baixo nível, nomeadamente no desenho de sistemas de operação [16].

Os tipos de sessão providenciam um mecanismo para especificar e raciocinar sobre protocolos de interacção num cenário cliente/servidor. É então natural que os encontremos na base de várias propostas para a modelação e análise de sistemas orientados a serviços (por exemplo [8]). Contudo, abordagens baseadas na noção tradicional de sessão não são capazes de endereçar interacção entre múltiplos participantes, pelo menos não de uma forma tipificada, deixando cenários como colaborações em tarefas de serviço entre vários participantes fora do seu alcance. Para resolver este problema alguns trabalhos estenderam a noção clássica de sessão por forma a poderem capturar interacção entre vários participantes, nomeadamente as propostas de Honda et al. [20], de Bettini et al. [6] e de Bonelli et al. [7]. Contudo, tais abordagens baseiam-se em configurações estáticas e não parecem adequadas para representar colaborações de serviço de natureza dinâmica.

Em particular, o trabalho proposto por Honda et al. [20] endereça o problema de verificar que todos os participantes de uma sessão com múltiplos participantes seguem um protocolo de interacção bem definido, baseando-se numa especificação do protocolo global (um tipo global) que descreve as interacções ponto a ponto entre participantes. Contudo, este tipo de especificação deixa pouco espaço para colaborações estabelecidas de forma dinâmica, pois os papéis de cada participante têm de ser conhecidos à partida. Esta abordagem é estendida na proposta de Bettini et al. [6] mas não na direcção de capturar sistemas dinâmicos. Em vez disso, o foco é posto no problema difícil de garantir ausência de pontos de bloqueio ("deadlock absence") em sistemas onde as várias partes estão envolvidas em múltiplas sessões entre múltiplos participantes, baseando-se numa noção de ordenação dos meios de comunicação. Contudo, a técnica apresentada apresenta algumas limitações ao nível de capturar sistemas em que as partes adquirem acesso de forma dinâmica aos meios de comunicação. Estes problemas em aberto formam um conjunto de perguntas, para as quais o nosso trabalho apresenta contribuições relevantes, nomeadamente:

- Como é possível garantir que interacções entre participantes distribuídos seguem protocolos de interacção bem definidos, não havendo controlo central?
- Como é que asseguramos tal propriedade, mesmo em cenários onde participantes que não estavam previstos à partida se juntam às colaborações de forma dinâmica?
- Como é que certificamos que sistemas onde as várias partes estão envolvidas em múltiplos protocolos simultaneamente estão livres de pontos de bloqueio?
- Como é que verificamos estas propriedades, olhando só para o código fonte dos programas?
- Como é que conseguimos especificar os programas de forma rigorosa tendo em vista a sua análise através de mecanismos automáticos e demonstravelmente correctos?

Na secção seguinte apresentamos as principais ideias que estão na base da nossa abordagem e de que forma é que estas permitem contribuir para esta lista de problemas.

3 Modelação e Análise de Sistemas Orientados a Serviços

Enquanto grande parte das aplicações orientadas a serviços mais tradicionais podem ser modeladas no paradigma cliente/servidor, dado que os serviços providenciados a um cliente final frequentemente enquadram-se nesta descrição, este não parece ser o caso para aplicações orientadas a serviços em geral. Por exemplo, se considerarmos um serviço que implementa um inteiro processo de negócio que tem de simultaneamente lidar com compras electrónicas, gestão de fornecedores, interfaces de serviço a clientes, entre outros, então seguramente encontramos colaborações com vários fornecedores de serviço numa só aplicação. Exemplos comuns incluem o recurso a parceiros externos para levar a cabo o re-fornecimento de produtos ou serviços de entrega ao cliente. Então, apesar de tipicamente uma única interacção envolver apenas duas partes, um conjunto de interacções relacionadas por pertencerem à mesma tarefa de serviço podem envolver várias partes. Claramente, para que uma tarefa de serviço seja realizada com sucesso é necessário que os vários parceiros envolvidos tenham um entendimento comum de qual é o protocolo global de interacção.

Os ingredientes básicos de um sistema orientado a serviços são então um número de aplicações que estão distribuídas na rede que interagem por forma a colaborarem numa tarefa. Um aspecto chave na organização destes sistemas é então a noção de um conjunto de interacções relacionadas por pertencerem à mesma tarefa: uma *conversação*. Uma conversação é um conjunto estruturado de interacções entre várias partes, sem controlo central e possivelmente concorrente. A nossa noção de conversação pode ser vista como uma generalização da noção de sessão. Elaboramos este ponto de seguida, através de uma comparação mais detalhada.

3.1 Sessões Binárias versus Conversações

As sessões binárias, mencionadas na secção 2.2, capturam a interacção entre dois participantes, tipicamente um cliente e um servidor, que trocam mensagens de forma sequencial e simétrica: se um está pronto para enviar uma mensagem então o outro está pronto para a receber e vice-versa. As sessões providenciam um meio para encapsular um conjunto de interacções relacionadas, servindo então como um mecanismo simples para especificar e analisar interacção entre processos.

Por exemplo, consideremos a troca de mensagens ilustrada na Figura 1 que representa uma interacção de serviço. Um cliente pretende comprar um item a um vendedor usando para tal o serviço intitulado *ServiçoCompras*. A interacção começa pela invocação por parte do cliente do *ServiçoCompras*. Depois, o cliente envia uma mensagem `compra` onde especifica o item que tenciona comprar, após a recepção da qual o vendedor responde ao cliente na mensagem `preço` que contém o preço do artigo. As mensagens `compra` e `preço` estão relacionadas pela mesma interacção de serviço, instanciada no momento da invocação do serviço. Este cenário pode ser modelado por uma sessão: a interacção começa pela criação de um meio de comunicação partilhado pelas duas partes, após o qual podem ser trocadas mensagens no mesmo.

A noção de sessão é então particularmente adequada para capturar o paradigma cliente/servidor, mas fica aquém quando se trata de capturar a interacção entre vários participantes, um cenário frequentemente encontrado em aplicações orientadas a serviços reais. A única maneira de envolver vários participantes numa sessão é através da delegação da participação: um participante de uma sessão pode pedir a outra entidade que o substitua na sessão, mas enquanto esta outra entidade ganha acesso à sessão, o participante que delega perde completamente o acesso à sessão. Donde, este mecanismo de delegação não basta para modelar interacção entre múltiplas partes. Contudo, podemos perguntar se bastaria uma noção mais refinada de delegação para dar suporte a interac-

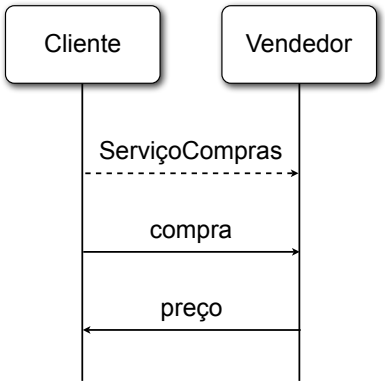

Figura 1: Interacção Cliente Vendedor.

ções entre múltiplos participantes: então e se os participantes que dão acesso a outros pudessem continuar a comunicar no meio de comunicação delegado?

Consideremos a extensão do exemplo anterior ilustrada na Figura 2, que mostra uma interacção típica num sistema orientado a serviços (adaptada de [11]). Tal como no primeiro exemplo, o cliente e o vendedor começam uma colaboração de serviço através do *ServiçoCompras* e trocam as mensagens `compra` e `preço`. Depois disso, o vendedor pede a um distribuidor para se juntar à colaboração que está a decorrer, através da invocação do *ServiçoEntregas*. Nessa altura, o vendedor informa o distribuidor qual é o artigo a ser entregue na mensagem `artigo`. De referir que, apesar do vendedor ter pedido ao distribuidor para se juntar à colaboração, o vendedor continua a participar na colaboração. Finalmente, o distribuidor transmite directamente ao cliente os detalhes da entrega através do envio da mensagem `detalhes`. A conversação de serviço como um todo consiste então na troca das mensagens `compra`, `preço`, `artigo` e `detalhes` entre os três participantes cliente, vendedor e distribuidor.

Este cenário pode ser modelado pela nossa noção de conversação: para começar, um meio de comunicação é criado e partilhado entre cliente e vendedor, onde são trocadas mensagens entre as duas partes, tal como numa sessão. Posteriormente, o distribuidor junta-se à colaboração de serviço e é-lhe dado acesso ao meio de comunicação partilhado. Assim, o acesso ao meio de comunicação pode ser cedido de forma dinâmica a outros, sem que o participante que delega perca o acesso, algo que pode ser descrito como uma forma de delegação parcial. Esta característica distingue as nossas conversações de qualquer proposta existente baseada em sessões, permitindo-nos endereçar cenários complexos de colaborações de serviço que envolvem múltiplos participantes determinados dinamicamente.

Podemos descrever a relação entre conversações e sessões da seguinte forma: as conversações são uma simples extensão da noção de sessão que suportam interacção entre múltiplos participantes num único meio de comunicação dedicado. As conversações, tais como as sessões, providenciam um meio para encapsular um conjunto de interacções relacionadas. Contudo, contrariamente às sessões, as interacções podem tomar forma entre múltiplas partes do sistema. Tal como nas sessões, uma única interacção numa conversação é binária no sentido que uma mensagem é trocada apenas entre dois participantes. Porém, contrariamente às sessões, vários participantes podem estar interagir concorrentemente, trocando mensagens distintas. As conversações e as sessões são inicialmente estabelecidas entre duas partes. Contudo, as conversações, contrariamente às sessões, permitem que outros participantes se juntem (ou saiam) de forma dinâmica, o que faz com que o número de participantes numa conversação possa dinamicamente crescer (ou decrescer), o que corresponde aos cenários mais realistas, quando pensamos em sistemas baseados na Web e

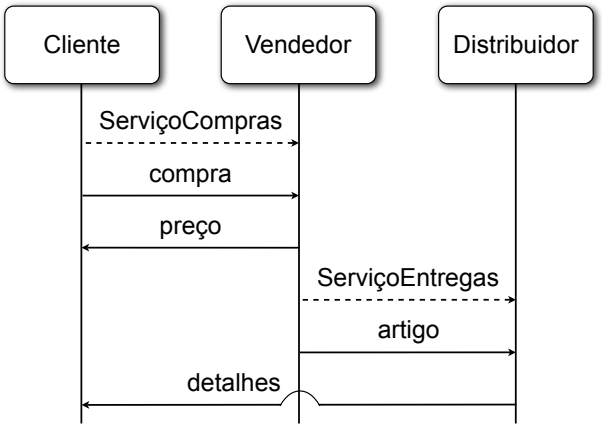

Figura 2: Interacção Cliente Vendedor Distribuidor.

computação na "cloud".

Esta noção básica da nossa noção de conversação permite-nos desde já ver a sua utilidade para modelar interacção entre múltiplos participantes numa colaboração de serviço. Antes de entrar em mais detalhe em como é que instanciamos esta noção num modelo concreto, voltemos um pouco atrás para descrever quais são, de um ponto de vista abstracto, as características principais da computação orientada a serviços que são endereçados no nosso trabalho.

3.2 Aspectos Chave da Computação Orientada a Serviços

Identificamos os seguintes aspectos chave da computação orientada a serviços: *distribuição*, *delegação* de processos, *contextos* de comunicação e *acoplamento fraco*.

3.2.1 Distribuição

Do ponto de vista de um cliente de um serviço, a principal motivação para invocar um serviço é conseguir que outros realizem uma tarefa em seu lugar. Isto deixa o cliente do serviço livre para executar outras tarefas, permitindo distribuir carga de trabalho ou suportar noções de especialização de tarefas em colaborações de serviço: um parceiro pode concentrar-se em tarefas para as quais pode mais facilmente proporcionar soluções e delegar outras tarefas para outros serviços. Então, o objectivo de uma chamada de serviço é a delegação de uma tarefa para um parceiro que usará os seus próprios recursos remotos para levar a cabo essa tarefa. Isto implica que a relação entre quem providencia o serviço e quem o chama faz particularmente sentido quando estes são entidades distintas, com acesso a diferentes recursos e capacidades, tipicamente a executar em máquinas diferentes, donde quando exista uma noção subjacente de sistema distribuído.

Um aspecto essencial que já identificámos é então a noção de delegação remota de processos. Esta noção difere de noções mais tradicionais como invocação remota de operações, que encontramos em sistemas de objectos distribuídos, ou mesmo de invocação remota de procedimentos em sistemas cliente/servidor.

3.2.2 Delegação de Processos versus Invocação de Operações

Num sistema distribuído o único mecanismo de comunicação verdadeiramente suportado é a troca de mensagens, o que leva a um modelo assíncrono de interacção. Por cima deste mecanismo básico podem-se representar abstracções mais sofisticadas, nomeadamente invocação remota de

procedimentos e invocação remota de métodos, que são usadas de uma forma chamada/retorno. Por sua vez, uma invocação de serviço implica a execução remota de uma tarefa complexa que pode requisitar um padrão de interacção mais rico, que pode na realidade envolver várias partes. Em geral, um cliente de serviço delega toda uma actividade interactiva (tecnicamente, um processo) para o fornecedor do serviço. Desta forma, vemos a invocação de serviço a um nível de abstracção mais elevado em comparação com a invocação remota de procedimentos e a invocação remota de métodos.

Como exemplos típicos de sub-actividades que podem ser delegadas para parceiros externos podemos referir um serviço de marcação de voos (numa agência de viagens), um serviço de aquisição de artigos (num departamento de compras), um serviço que implemente movimentos bancários (num multibanco), um serviço que guarde e busque documentos (num arquivo), um serviço que receba e envie correio (num departamento de expedição), entre outros. Na nossa perspectiva, um aspecto que distingue a computação orientada a serviços é então esta ênfase na delegação remota de processos, em vez da delegação remota de operações individuais.

O paradigma da delegação remota de processos parece mais geral que a delegação remota de operações, pelo menos a este nível de descrição, pois podemos sempre considerar uma operação individual como um caso particular de um processo interactivo. Por outro lado, podemos também dizer que a delegação de um inteiro processo interactivo pode ser implementada, a um mais baixo nível, através da invocação de várias operações individuais. Contudo, estas operações remotas individuais estão de qualquer das formas contextualmente relacionadas por pertencerem à mesma tarefa e provavelmente têm de transportar informação que explícita ou implicitamente as ligue. Parece então que uma abstracção de meio de comunicação adequada para a computação orientada a serviços deverá ser capaz de acomodar um conjunto de interacções relacionadas.

3.2.3 Contextos de Comunicação

Um contexto é um espaço onde computações e comunicações acontecem. Um contexto pode ter um significado espacial, como por exemplo uma localização num sistema distribuído, mas também um significado comportamental, como por exemplo um contexto de conversação, ou uma sessão, entre dois ou mais participantes. Por exemplo, a mesma mensagem pode aparecer em dois contextos distintos, com significados diferentes. Na prática verificamos que tais mensagens transportam informação que as liga de forma inequívoca a alguma informação contextual (na tecnologia de serviços Web as mensagens são etiquetadas com "correlation tokens" que servem para identificar a conversação de serviço à qual as mensagens pertencem). Ter uma representação explícita destes contextos parece então um mecanismo conveniente para estruturar e encapsular conversações de serviço.

Um contexto é também uma abstracção natural para agrupar e publicar um conjunto de serviços que estejam relacionados. Tipicamente, os serviços publicados pela mesma entidade provavelmente partilham recursos, sendo que esta partilha pode ser observada a diversos níveis de granularidade. Exemplos extremos são um objecto (da programação orientada a objectos), onde as definições de serviços são métodos e o contexto partilhado é o estado interno do objecto, ou uma entidade como por exemplo a Amazon, que publica vários serviços para variadas funcionalidades, onde certamente existirá partilha de recursos, desde bases de dados a serviços internos de cobrança, entre outros.

Por um lado, os contextos servem para agrupar um conjunto de interacções relacionadas e para agrupar recursos que são partilhados por uma parte do sistema. Por outro lado, os contextos induzem fronteiras entre computação e comunicação não fortemente ligadas. Tais fronteiras são essenciais para dar suporte a uma noção de acoplamento fraco entre as várias partes de um sistema.

3.2.4 Acoplamento Fraco

Uma aplicação baseada em serviços normalmente consiste numa colecção de instâncias de serviços parceiros, aos quais irão ser delegadas determinadas tarefas, alguns processos que executam localmente e um ou mais processos de controlo (ou orquestração). A flexibilidade e abertura de um desenho baseado em serviços, ou pelo menos uma característica desejada, resulta do fraco acoplamento entre estes vários ingredientes. Por exemplo, uma orquestração que descreve um processo de negócio deve ser feita de uma forma bastante independente de quais são em concreto as instâncias de serviço auxiliares, desta forma permitindo que a execução das tarefas possa ser levada a cabo por serviços descobertos dinamicamente. Na linguagem de descrição de serviços Web WSDL [13], o acoplamento fraco é de alguma forma conseguido através da declaração separada de um nível abstracto, que descreve um serviço pelas mensagens que este recebe e envia (um interface comportamental), e de um nível concreto que especifica qual é a informação de mais baixo nível, como por exemplo informação de transporte das mensagens na rede. Desta forma, serviços que partilham o mesmo interface abstracto podem ser referidos de forma genérica, independentemente dos detalhes de comunicação de mais baixo nível.

Para evitar acoplamento forte de serviços, a interface entre uma instância de serviço e o contexto de instanciação deve ser feita através de processos mediadores apropriados, de forma a esconder e/ou adaptar os protocolos de comunicação do serviço (que dependem, por exemplo, de qual é o fornecedor de serviço usado) de/para o interface comportamental esperado pelo contexto de instanciação. Todas as entidades que cooperam numa tarefa de serviço devem então ser encapsuladas (delimitadas por um contexto de conversação) e capazes de comunicar entre elas e com o contexto externo apenas através de um mecanismo genérico de troca de mensagens.

3.2.5 Outros Aspectos

Focamos o nosso desenvolvimento neste conjunto de características chave da computação orientada a serviços, deixando de fora do escopo da nossa abordagem aspectos que devem ser endereçados num modelo completo para computação orientada a serviços. Os mais óbvios incluem tratamento de falhas, segurança (em particular, controlo de acesso e autenticação) e um mecanismo de interoperação. Este último parece especialmente relevante e certamente sugere um critério de avaliação importante para qualquer modelo para computação orientada a serviços.

Dada esta caracterização geral do cenário que pretendemos cobrir, podemos agora apresentar a nosso modelo para computação orientada a serviços.

3.3 O Cálculo das Conversações

Nesta secção apresentamos a nossa proposta de modelo para a computação orientada a serviços, o Cálculo das Conversações (CC) [32]. Começamos por apresentar as primitivas individualmente e como é que estas instanciam os aspectos mencionados anteriormente.

O CC pode ser visto como uma especialização do Cálculo-π para computação orientada a serviços. O CC inclui os operadores básicos de estruturação de processos (existentes no Cálculo-π), nomeadamente a composição paralela de processos $P \mid Q$ que representa que os processos P e Q estão a executar em simultâneo, o operador de escolha $P + Q$ que representa um processo que ou executa P ou executa Q, o processo inactivo $\mathbf{0}$ e o operador de restrição de nomes $(\boldsymbol{\nu} a)P$ que especifica que o nome a é local ao processo P. Depois, o CC inclui também primitivas específicas à computação orientada a serviços que apresentamos de seguida.

3.3.1 Acesso a Contexto de Conversação

Um contexto de conversação é um meio onde interacções relacionadas têm lugar. Cada contexto é identificado por um nome único (cf., um URI). Então, para interagir numa conversação um processo precisa apenas saber o nome da mesma por forma a aceder ao respectivo contexto de conversação. Isto permite aos processos aceder às conversações a partir de qualquer ponto no sistema. Denotamos por $c \blacktriangleleft [P]$ um processo que acede à conversação com nome c e que interage nessa conversação de acordo com o que é especificado em P. Um contexto de conversação pode na realidade estar distribuído em vários pontos de acesso, sendo que processos localizados nesses vários pontos de acesso interagem entre eles de forma transparente à distribuição. Intuitivamente, um contexto de conversação pode ser descrito como uma sala de "chat" virtual onde vários participantes remotos trocam mensagens, podendo estar envolvidos simultaneamente em várias conversações.

Potencialmente, cada ponto de acesso vai ser colocado dentro de um contexto de execução distinto. Por outro lado, cada ponto de acesso é necessariamente colocado num único contexto de execução. A relação entre contexto envolvente e um ponto de acesso pode ser comparada com a de chamada/chamador, mas onde ambas as entidades podem continuamente interagir entre si.

3.3.2 Percepção de Contexto

Um processo que esteja a interagir num determinado contexto de conversação pode adquirir de forma dinâmica a identidade dessa conversação. Esta capacidade é suportada pela primitiva de percepção de contexto ("context awareness") $\textbf{this}(x).P$. A variável x vai ser instanciada pelo nome da conversação corrente. Por exemplo, o processo $c \blacktriangleleft [\ \textbf{this}(x).P\ ]$ evolui para $c \blacktriangleleft [\ P\{x/c\}\ ]$ num passo de execução, onde $\{x/c\}$ representa a substituição de todas as ocorrências livres da variável x pelo nome c. Esta primitiva tem algumas semelhanças com o **self** (ou **this**) das linguagens orientadas a objectos, mesmo tendo uma semântica distinta.

3.3.3 Comunicação

A comunicação entre subsistemas é realizada por trocas de mensagens. Visto que várias mensagens podem ser concorrentemente trocadas estas são etiquetadas com identificadores (por exemplo, `compra`) para que se possam diferenciar. Em primeiro lugar, denotamos a recepção e o envio de mensagens de/para a conversação corrente através dos operadores: $\texttt{etiqueta}^{\downarrow}?(x_1, \ldots, x_n).P$ e $\texttt{etiqueta}^{\downarrow}!(v_1, \ldots, v_n).P$. No caso do envio, os termos v_i representam os argumentos da mensagem, os valores a serem enviados. No caso da recepção, as variáveis x_i representam os parâmetros das mensagens que estão ligados em P. O símbolo $\downarrow$ (ler "aqui") diz que as acções de comunicação devem ser exercidas na conversação corrente.

Em segundo lugar, denotamos a recepção e o envio de mensagens de/para a conversação envolvente através dos operadores: $\texttt{etiqueta}^{\uparrow}?(x_1, \ldots, x_n).P$ e $\texttt{etiqueta}^{\uparrow}!(v_1, \ldots, v_n).P$. O símbolo $\uparrow$ (ler "cima") diz que as acções de comunicação devem ser exercidas na (univocamente determinada) conversação envolvente, sendo a envolvência determinada pelo contexto onde o processo que exerce tais capacidades de comunicação está a correr.

3.3.4 Idiomas de Primitivas Orientadas a Serviços

Apesar de não os considerarmos como nativos à linguagem (pois podem ser expressos através dos mecanismos de comunicação básicos), é desde já útil introduzir alguns idiomas dedicados à publicação e instanciação de serviços, assim como um idioma que permite que serviços externos se juntem a uma conversação enquanto esta está a decorrer. Um contexto (que usamos de forma

idiomática para representar uma localização ou uma entidade) pode publicar uma ou mais definições de serviço. As definições de serviço são entidades sem estado, comparáveis a funções numa linguagem de programação funcional. A definição de um serviço é expressa pelo idioma:

$$\textbf{def}\ \texttt{NomeServiço} \Rightarrow \textit{CorpoServiço}$$

onde `NomeServiço` é o nome do serviço e *CorpoServiço* é o processo que deve ser executado pelo fornecedor do serviço quando este é instanciado. Para poder ser invocada, uma definição de serviço deve ser inserida num contexto, por exemplo:

$$\textit{FornecedorServiço} \blacktriangleleft [\ \textbf{def}\ \texttt{NomeServiço} \Rightarrow \textit{CorpoServiço}\ \cdots\ ]$$

Este serviço pode ser instanciado usando o seguinte idioma:

$$\textbf{new}\ \textit{FornecedorServiço} \cdot \texttt{NomeServiço} \Leftarrow \textit{ProtocoloCliente}$$

onde *FornecedorServiço* é o nome do contexto onde o serviço `NomeServiço` está definido e onde *ProtocoloCliente* identifica o processo que vai ser executado pelo cliente. O resultado de uma instanciação de serviço é a criação de uma nova identidade de uma conversação (globalmente nova — um nome restrito) e a criação de dois pontos de acesso para essa mesma conversação. Um dos pontos de acesso irá conter o *CorpoServiço* e vai estar localizada dentro do contexto *FornecedorServiço*. O outro vai conter o *ProtocoloCliente* e vai estar localizado no mesmo contexto da expressão **new** que instanciou o serviço. Estes novos contextos de conversação apresentam-se nos contextos "chamadores" como qualquer outro processo local, pois os processos *CorpoServiço* e *ProtocoloCliente* podem interagir continuamente no contexto chamador através de mensagens dirigidas para $\uparrow$. Por sua vez, interacções entre *CorpoServiço* e *ProtocoloCliente* na nova conversação são realizadas através de mensagens dirigidas para $\downarrow$.

Primitivas como estas que permitem a publicação e instanciação de serviços podem ser encontradas em abordagens baseadas em sessões. Contudo, na nossa abordagem estas primitivas não são nativas ao modelo, estando suportadas por mecanismos de comunicação mais canónicos. Por outro lado, o idioma seguinte é novo à nossa abordagem. Aliás, cremos que não é possível representá-lo em abordagens anteriores baseadas em sessões. No nosso modelo os identificadores de conversação podem ser manipulados pelos processos se necessário, acedidos através da primitiva **this**, passados em trocas de mensagens e estão sujeitos a extrusões de escopo: isto permite-nos modelar conversações entre múltiplos participantes pelo acesso progressivo de múltiplas entidades, possivelmente determinadas dinamicamente, a uma conversação que está a decorrer. O juntar de uma entidade externa a uma conversação que está a decorrer é um padrão de programação frequente que pode ser abstraído pelo idioma:

$$\textbf{join}\ \textit{FornecedorServiço} \cdot \textit{NomeServiço} \Leftarrow \textit{ProcessoContinuação}$$

A semântica da expressão **join** é semelhante à do idioma de instanciação de serviço **new**: a diferença chave é que enquanto o **new** cria uma *nova* conversação, o **join** permite que o *NomeServiço* publicado em *FornecedorServiço* se junte à conversação corrente e o processo que executa o **join** continua a interagir na conversação corrente tal como especificado em *ProcessoContinuação*.

3.4 Sintaxe e Semântica do Cálculo das Conversações

Nesta secção definimos a sintaxe e a semântica do modelo que propomos para a modelação de sistemas orientados a serviços. Apresentamos as principais definições e resultados, omitindo alguns detalhes técnicos, tendo apenas por objectivo dar ao leitor uma visão geral dos formalismos envolvidos. Depois de apresentada a sintaxe introduzimos uma interpretação matemática para as

$$
\begin{array}{lcll}
P, Q, R & ::= & \mathbf{0} & \text{(Inacção)} \\
 & | & P \mid Q & \text{(Composição Paralela)} \\
 & | & (\boldsymbol{\nu} a)P & \text{(Restrição de Nomes)} \\
 & | & \textbf{rec}\, \mathcal{X}.P & \text{(Recursão)} \\
 & | & \mathcal{X} & \text{(Variável)} \\
 & | & n \blacktriangleleft [P] & \text{(Acesso à Conversação)} \\
 & | & \Sigma_{i \in I}\, \alpha_i.P_i & \text{(Escolha Guardada por Prefixos)} \\
 & & & \\
\alpha & ::= & l!(n) & \text{(Envio)} \\
 & | & l?(x) & \text{(Recepção)} \\
 & | & \textbf{this}(x) & \text{(Percepção de Contexto)}
\end{array}
$$

Figura 3: Sintaxe do Cálculo das Conversações.

várias construções da linguagem, a semântica operacional. Depois concretizamos a noção de objecto semântico através de uma equivalência comportamental e reportamos alguns resultados que mostram que o modelo goza de algumas propriedades teóricas essenciais. Para simplificar a apresentação consideramos o fragmento monádico do cálculo (os tuplos comunicados são de dimensão 1) e sem direcções nas mensagens (todas as mensagens são relativas à conversação corrente).

A sintaxe do Cálculo das Conversações é dada pela Figura 3, onde assumimos dado um conjunto infinito de nomes ($a, b, c, \ldots \in \Lambda$), um conjunto infinito de variáveis ($x, y, z, \ldots \in \mathcal{V}$), um conjunto infinito de etiquetas ($l, s \ldots \in \mathcal{L}$) e um conjunto infinito de variáveis de recursão ($\mathcal{X}, \mathcal{Y}, \ldots \in \chi$). Usamos ainda um conjunto de identificadores para referir nomes ou variáveis ($n, m, o, \ldots\ldots \in \Lambda \cup \mathcal{V}$). Descrevemos informalmente as várias primitivas de forma resumida, dada a apresentação que lhes foi feita na secção anterior: a inacção **0** representa o processo inactivo ou terminado; a composição paralela $P \mid Q$ representa que os processos P e Q estão a executar simultaneamente; a restrição de nomes $(\boldsymbol{\nu} a)P$ representa que o nome a é local ao processo P; e a recursão **rec** $\mathcal{X}.P$ representa um processo que executa P repetidamente. Estas primitivas definem o fragmento estático do Cálculo das Conversações e também se encontram na base do Cálculo-π. Uma primitiva específica à computação orientada a serviços introduzida no Cálculo das Conversações é o acesso à conversação $n \blacktriangleleft [P]$, que representa um processo que interage na conversação n de acordo com o especificado em P.

A comunicação é expressa através da escolha guardada por prefixos $\Sigma_{i \in I}\, \alpha_i.P_i$, que representa um processo que pode realizar qualquer uma das acções iniciais α_k e depois prosseguir como especificado na respectiva continuação P_k. As acções de comunicação são de dois tipos: envio de mensagem $l!(n)$ onde l é a etiqueta da mensagem e n é o nome a ser enviado e recepção de mensagem $l?(x)$ onde l é a etiqueta da mensagem e x é a variável que irá ser instanciada pelos valor recebido. Uma acção pode também ser a percepção de contexto **this**(x), que permite a um processo aceder à identidade da conversação corrente de forma dinâmica.

A semântica operacional do Cálculo das Conversações está definida através de um sistema de transições etiquetadas. Uma transição $P \xrightarrow{\lambda} Q$ especifica que o processo P pode evoluir para o processo Q através da realização da acção representada por λ. As etiquetas de transição e as acções são dadas na Figura 4. A acção τ representa uma interacção interna, enquanto que as acções $l!(a)$ e $l?(a)$ representam interacções com o ambiente exterior. Por sua vez, a acção `this` representa um acesso à identidade da conversação, capturando o comportamento de um processo que pretende aceder à identidade da conversação (através da primitiva **this**), permitindo capturar também comportamentos que dependem dessa informação contextual para terem lugar.

Para capturar a semântica observacional dos processos, as etiquetas de transição têm de espe-

$$\begin{array}{lcll} \sigma & ::= & \tau \mid l!(a) \mid l?(a) \mid \texttt{this} & \text{(Acções)} \\ \lambda & ::= & c\,\sigma \mid \sigma \mid (\nu a)\lambda & \text{(Etiquetas de Transição)} \end{array}$$

Figura 4: Etiquetas de Transição e Acções.

$$l!(a).P \xrightarrow{l!(a)} P \text{ (\textit{Envio})} \qquad l?(x).P \xrightarrow{l?(a)} P\{x/a\} \text{ (\textit{Recepção})} \qquad \frac{\alpha_j.P_j \xrightarrow{\lambda} Q \quad j \in I}{\Sigma_{i\in I}\,\alpha_i.P_i \xrightarrow{\lambda} Q}(\textit{Soma})$$

$$\frac{P \xrightarrow{\lambda} Q \quad a \in \mathit{out}(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{(\nu a)\lambda} Q}(\textit{Abertura}) \qquad \frac{P \xrightarrow{\lambda} Q \quad a \notin \mathit{na}(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{\lambda} (\boldsymbol{\nu} a)Q}(\textit{Restrição})$$

$$\frac{P \xrightarrow{\lambda} Q \quad \mathit{bn}(\lambda)\,\#\mathit{fn}(R)}{P \mid R \xrightarrow{\lambda} Q \mid R}(\textit{Paralelo}) \qquad \frac{P\{\mathcal{X}/\textbf{rec}\,\mathcal{X}.P\} \xrightarrow{\lambda} Q}{\textbf{rec}\,\mathcal{X}.P \xrightarrow{\lambda} Q}(\textit{Recursão})$$

$$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}(\textit{Comunicação}) \qquad \frac{P \xrightarrow{(\nu a)\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad a \notin \mathit{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} a)(P' \mid Q')}(\textit{Fecho})$$

Figura 5: Operadores Básicos.

cificar não só a acção mas também a conversação à qual a acção diz respeito. Assim, dizemos que uma etiqueta de transição da forma $c\ \sigma$ está localizada na conversação c ou não localizada caso contrário. A etiqueta $(\nu a)\lambda$ especifica que o nome a é emitido de forma ligada em λ, ou seja é um nome local (restrito) a um processo que está a ser comunicado para o ambiente exterior. Dada uma etiqueta de transição λ, denotamos por $\overline{\lambda}$ a etiqueta dual obtida através da troca das acções de envio por recepção e vice-versa (e.g., $\overline{l!(a)} = l?(a)$ e $\overline{l?(a)} = l!(a)$).

Podemos agora definir o sistema de transições etiquetado. Dividimos a apresentação das regras em dois conjuntos para facilitar a sua leitura: as regras dos operadores básicos apresentadas na Figura 5 seguem as linhas dos sistemas de transições etiquetadas do Cálculo-π; por outro lado, as regras dos operadores de conversação apresentadas na Figura 6 são específicas ao nosso modelo.

Descrevemos informalmente as regras apresentadas na Figura 5. Na regra (*Envio*) é observada uma emissão no respectivo prefixo de comunicação, onde o processo de chegada corresponde à continuação do prefixo. De forma análoga na regra (*Recepção*) é observada uma recepção, correspondendo o processo chegada à continuação do prefixo onde todas as ocorrências livres da variável são substituídas pelo nome recebido ($\{x/a\}$). Na regra (*Soma*) é observada uma transição que selecciona um dos prefixos iniciais da escolha. Na regra (*Abertura*) o escopo do nome restrito é aberto pois o nome é emitido na mensagem ($a \in \mathit{out}(\lambda)$). Por outro lado, quando a acção observada não menciona o nome restrito então esta passa de forma transparente através da restrição de nomes — regra (*Restrição*). A regra (*Paralelo*) especifica que as acções dos ramos paralelos são observadas a nível da composição paralela, garantindo-se que os nomes ligados da acção não colidem com os nomes livres do processo ($\mathit{bn}(\lambda)\,\#\mathit{fn}(R)$). A regra (*Recursão*) especifica que o processo recursivo exibe as mesmas transições do processo obtido pelo desdobramento da definição recursiva. As duas últimas regras da Figura 5 capturam a sincronização entre processos: a regra (*Comunicação*) especifica que se os dois ramos de uma composição paralela exibem acções duais então estes processos podem sincronizar, sendo observada uma acção interna τ. De forma semelhante a regra (*Fecho*) captura a sincronização entre dois processos paralelos em acções duais,

$$\frac{P \xrightarrow{\lambda} Q \quad unloc(\lambda)}{c \blacktriangleleft [P] \xrightarrow{c\cdot\lambda} c \blacktriangleleft [Q]}(\textit{NãoLoc}) \qquad \frac{P \xrightarrow{a\ \lambda} Q \quad \lambda \neq \texttt{this}}{c \blacktriangleleft [P] \xrightarrow{a\ \lambda} c \blacktriangleleft [Q]}(\textit{Loc}) \qquad \frac{P \xrightarrow{\tau} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]}(\textit{Tau})$$

$$\textbf{this}(x).P \xrightarrow{c\ \texttt{this}} P\{x/c\}\ (\textit{LerCorrente}) \qquad \frac{P \xrightarrow{c\ \texttt{this}} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]}(\textit{LerId})$$

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \texttt{this}} P' \mid Q'}(\textit{ComLerId}) \qquad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu a)c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \texttt{this}} (\boldsymbol{\nu} a)(P' \mid Q')}(\textit{FechoLerId})$$

Figura 6: Operadores de Conversação.

mas onde é também comunicado um nome restrito. O escopo do nome restrito é então fechado contendo o processo que emitiu o nome e o processo que o recebe.

Descrevemos também as regras apresentadas na Figura 6. Na regra (*NãoLoc*) uma acção é localizada na conversação à qual diz respeito. Na regra (*Loc*) uma acção já localizada passa de forma transparente pela construção de acesso à conversação. Da mesma forma na regra (*Tau*) uma acção interna passa transparentemente pela fronteira do acesso à conversação. Na regra (*LerCorrente*) o prefixo **this** origina uma leitura à identidade da conversação corrente, capturada pela acção $c\ \texttt{this}$. Esta acção só pode progredir se a identidade da conversação corrente for de facto c, tal como é expresso pela regra (*LerId*). Por fim, as regras (*ComLerId*) e (*FechoLerId*) tratam a sincronização entre uma acção localizada e outra ainda não localizada: se a conversação corrente for a mesma da especificada na acção localizada então ambas as acções são referentes à mesma conversação e podem sincronizar — a identidade da conversação corrente é lida através da acção $c\ \texttt{this}$.

O sistema de transições etiquetadas descreve tanto as interacções internas como as interacções com o ambiente exterior. Quando nos focamos em sistemas fechados, que não estão sujeitos à interacção com o ambiente exterior, interessam-nos apenas os comportamentos internos e autónomos que correspondem às transições τ. Esta noção é capturada pela relação de redução: dizemos que P reduz para Q, denotado por $P \rightarrow Q$, se $P \xrightarrow{\tau} Q$.

Baseando-nos nas observações que podemos realizar sobre os processos, capturadas pelo sistema de transições etiquetadas, podemos agora definir a semântica comportamental do Cálculo das Conversações através de uma noção de equivalência comportamental. Usamos a noção standard de bisimilaridade, para definir a qual primeiro introduzimos a noção de bisimulação.

Definição 3.1 (Bisimulação)
Uma bisimulação é uma relação binária simétrica $\mathcal{R}$ sobre processos tal que, para todos os processos P e Q, se $P\,\mathcal{R}\,Q$ então:

Se $P \xrightarrow{\lambda} P'$ e $bn(\lambda)\,\#\,fn(Q)$
então existe Q' tal que $Q \xrightarrow{\lambda} Q'$ e $P'\,\mathcal{R}\,Q'$.

Essencialmente, dois processos estão relacionados por uma bisimulação se podemos fazer um jogo de observações simétrico onde as transições realizadas por um dos processos são imitadas pelo outro processo, levando a estados equivalentes. Provamos que as bisimulações são relações fechadas pela união, o que nos permite definir bisimilaridade como a união de todas as bisimulações.

Definição 3.2 (Bisimilaridade)
A relação de bisimilaridade, denotada por $\sim$, é a união de todas as bisimulações.

Provamos algumas propriedades básicas da relação de bisimilaridade, nomeadamente que se trata de uma relação de equivalência. Depois provamos que a relação de bisimilaridade é uma congruência para todas as construções da linguagem.

Teorema 3.3 *A relação de bisimilaridade é uma congruência.*

Este resultado garante-nos que os nossos operadores sintácticos correspondem a funções de comportamento bem definidas: dado um único comportamento abstracto descrito por dois processos, se colocarmos esses dois processos num contexto da linguagem igual então obtemos dois processos que descrevem o mesmo comportamento abstracto. Provamos também algumas propriedades comportamentais interessantes, tais como a validade da equação comportamental seguinte:

$$n \blacktriangleleft [P] \mid n \blacktriangleleft [Q] \sim n \blacktriangleleft [P \mid Q] \tag{1}$$

A equação (1) captura a noção do contexto de conversação ser um único meio de comunicação que pode ser acedido a partir de vários pontos.

3.5 Implementação dos Idiomas Orientados a Serviços

Nesta secção mostramos como são implementados os idiomas de serviço **def**, **new** e **join** usando as primitivas básicas da linguagem. Uma definição **def** $s \Rightarrow P$ publica na conversação corrente o serviço s, sendo implementada da seguinte forma:

$$\textbf{def}\ s \Rightarrow P \quad \triangleq \quad s?(x).x \blacktriangleleft [P]$$

Essencialmente, uma definição de serviço é uma recepção de mensagem (cuja etiqueta corresponde ao nome do serviço), correspondendo o nome recebido à identidade da conversação onde as interacções relativas a essa instância de serviço vão ter lugar — daí a colocação do corpo do serviço P num acesso à conversação x. Podemos definir um serviço persistente nas mesmas linhas usando a definição recursiva de processos. De recordar que as definições de serviço devem ser colocadas num contexto de conversação, por exemplo:

$$n \blacktriangleleft [\ \textbf{def}\ s \Rightarrow P\ ] \quad \triangleq \quad n \blacktriangleleft [\ s?(x).x \blacktriangleleft [P]\ ]$$

Para instanciar uma definição de serviço podemos usar o idioma **new** ou o idioma **join**, sendo que no primeiro instanciamos o serviço numa conversação nova enquanto que no segundo pedimos ao fornecedor de serviço que se junte a uma conversação que está a decorrer.

A implementação do idioma **new** é a seguinte:

$$\textbf{new}\ n \cdot s \Leftarrow Q \quad \triangleq \quad (\boldsymbol{\nu} c)(n \blacktriangleleft [s!(c)] \mid c \blacktriangleleft [Q])$$

onde é emitida uma identidade fresca (um nome restrito) na mensagem de serviço — trocada na conversação n onde o serviço está disponível — e é colocado o corpo Q num acesso à conversação cuja identidade é fresca. Por sua vez, o idioma **join** é implementado da seguinte forma:

$$\textbf{join}\ n \cdot s \Leftarrow Q \quad \triangleq \quad \textbf{this}(x).(n \blacktriangleleft [s!(x)] \mid Q)$$

onde na mensagem de serviço é emitida a identidade da conversação corrente (acedida através da primitiva **this**), continuando o processo Q a executar na conversação corrente.

Mostrámos como se consegue então capturar idiomas de programação de mais alto nível, usando apenas as primitivas canónicas de comunicação. Na secção seguinte analisamos a evolução de um pequeno programa por forma a dar alguma intuição de qual é o comportamento de um programa expresso no Cálculo das Conversações.

3.6 Um Programa Exemplo

Nesta secção apresentamos um exemplo de um programa CC, por forma a dar mais intuição sobre o significado das primitivas da linguagem descritas anteriormente. Aproveitamos a descrição preliminar feita na Secção 3.1 e implementamos o cenário aí descrito que envolve a compra de um artigo. Apesar de ser um exemplo simples e por isso mesmo permitir uma descrição mais detalhada, deixa desde logo uma ideia do estilo de programação suportado pela linguagem.

O cenário é composto por três participantes, um cliente, um vendedor e um distribuidor, onde o primeiro tenciona comprar um artigo ao segundo e o terceiro fica a cargo da entrega do artigo (na Figura 2 ilustramos a interacção entre estes três parceiros).

Para começar, de referir que modelamos de forma idiomática os três parceiros (cliente, vendedor e distribuidor) com contextos de conversação nomeados concordantemente. Temos então um sistema composto pela composição paralela de três contextos:

$$\mathit{Cliente} \blacktriangleleft [\,(\cdots)\,] \mid \mathit{Vendedor} \blacktriangleleft [\,(\cdots)\,] \mid \mathit{Distribuidor} \blacktriangleleft [\,(\cdots)\,]$$

O cliente começa por invocar o `ServiçoCompras` publicado por *Vendedor*, usando para tal o idioma **new** *Vendedor* $\cdot$ `ServiçoCompras` $\Leftarrow$ $(\cdots)$, onde o nome (`ServiçoCompras`) e o contexto onde o serviço está publicado (*Vendedor*) são identificados. Na instanciação do serviço é também especificada o protocolo de interacção seguido pelo cliente aquando da instanciação: primeiro envia a mensagem `compra`, depois recebe a mensagem `preço` e finalmente recebe a mensagem `detalhes`. O código CC que implementa a parte do cliente é então:

$$\textbf{new}\ \mathit{Vendedor} \cdot \texttt{ServiçoCompras} \Leftarrow \texttt{compra}^{\downarrow}!(\mathit{prod}).\texttt{preço}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}?(d)$$

onde *prod* é um valor que identifica o artigo (ou produto) que o cliente tenciona comprar (uma "string"), p é a variável que vai ser instanciada com o preço do artigo (um valor real) e d é a variável que vai ser instanciada com a informação da entrega (outra "string"). As mensagens, dirigidas para $\downarrow$, vão ser trocadas com os parceiros que colaboram na tarefa de serviço.

O vendedor publica o `ServiçoCompras` por via do idioma **def** `ServiçoCompras` $\Rightarrow$ $(\cdots)$. O protocolo de interacção do vendedor na colaboração de serviço é: primeiro recebe a mensagem `compra` e depois envia a mensagem `preço`. Pelo meio, vamos considerar que, para determinar o preço do artigo, o vendedor consulta uma base de dados local, acedida através de um processo interface que está a executar na conversação *Vendedor*. O primeiro bocado de código que implementa o papel do vendedor é então:

$$\texttt{compra}^{\downarrow}?(\mathit{prod}).\texttt{acedePreço}^{\uparrow}!(\mathit{prod}).\texttt{valorPreço}^{\uparrow}?(p).\texttt{preço}^{\downarrow}!(p).\ (\cdots)$$

Para interagir com a base de dados, o código do serviço acede ao contexto envolvente (o contexto *Vendedor*) através de mensagens dirigidas para $\uparrow$, enquanto que as mensagens que dizem respeito à conversação de serviço são dirigidas para $\downarrow$. Donde `compra` e `preço` são trocadas na conversação de serviço, enquanto que `acedePreço` e `valorPreço` são trocadas na conversação *Vendedor* com o processo interface da base de dados. Abstraímos da especificação concreta do processo interface da base de dados (*PreçoBD*), assumindo simplesmente que este está pronto para receber mensagens `acedePreço` (contendo a indicação do artigo em causa) respondendo às quais com uma mensagem `valorPreço` (que contém o valor).

Depois de enviar a mensagem `preço` o vendedor convida o distribuidor a juntar-se à conversação que está a decorrer. Esta funcionalidade pode ser realizada através do idioma **join** *Distribuidor*$\cdot$ `ServiçoEntregas` $\Leftarrow$ $(\cdots)$. Depois de pedir ao distribuidor para se juntar à conversação, o vendedor interage com o distribuidor, na conversação à qual o distribuidor acaba de se juntar, através da troca da mensagem `artigo`. Assim o código completo para o `ServiçoCompras` é:

$$\begin{aligned}\textbf{def}\ \texttt{ServiçoCompras} \Rightarrow\ & \texttt{compra}^{\downarrow}?(\mathit{prod}).\texttt{acedePreço}^{\uparrow}!(\mathit{prod}).\\ & \texttt{valorPreço}^{\uparrow}?(p).\texttt{preço}^{\downarrow}!(p).\\ & \textbf{join}\ \mathit{Distribuidor} \cdot \texttt{ServiçoEntregas} \Leftarrow \texttt{artigo}^{\downarrow}!(\mathit{prod})\end{aligned}$$

Finalmente, temos que o distribuidor publica o `ServiçoEntregas` onde o protocolo especificado é o seguinte: primeiro recebe a mensagem `artigo`, depois envia a mensagem `detalhes`.

$$\textbf{def}\ \texttt{ServiçoEntregas} \Rightarrow \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})$$

Colocando o código dos três intervenientes na colaboração de serviço nos respectivos contextos, juntamente com o processo interface da base de dados *PreçoBD* na conversação *Vendedor*, obtemos então a especificação completa do sistema:

$$\begin{array}{l}
\mathit{Cliente} \blacktriangleleft [\ \textbf{new}\ \mathit{Vendedor} \cdot \texttt{ServiçoCompras} \Leftarrow \texttt{compra}^{\downarrow}!(\mathit{prod}).\texttt{preço}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}?(d)\ ] \\
\mid \\
\mathit{Vendedor} \blacktriangleleft [\ \mathit{PreçoBD} \mid \\
\qquad \textbf{def}\ \texttt{ServiçoCompras} \Rightarrow \texttt{compra}^{\downarrow}?(\mathit{prod}).\texttt{acedePreço}^{\uparrow}!(\mathit{prod}). \\
\qquad\qquad\qquad\qquad\quad \texttt{valorPreço}^{\uparrow}?(p).\texttt{preço}^{\downarrow}!(p). \\
\qquad\qquad\qquad\qquad\quad \textbf{join}\ \mathit{Distribuidor} \cdot \texttt{ServiçoEntregas} \Leftarrow \texttt{artigo}^{\downarrow}!(\mathit{prod})\ ] \\
\mid \\
\mathit{Distribuidor} \blacktriangleleft [\ \textbf{def}\ \texttt{ServiçoEntregas} \Rightarrow \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})\ ]
\end{array}$$

Podemos agora ilustrar como é que este sistema evolui no tempo. Primeiro é instanciado o `ServiçoCompras` o que resulta na criação de uma conversação fresca (*conversaCompra*, por exemplo), sendo também estabelecidos dois pontos de acesso para esta conversação, um detido pelo cliente e o outro pelo vendedor. O sistema evolui então, num passo de execução, para:

$$\begin{array}{l}
(\nu \mathit{conversaCompra}) \\
(\mathit{Cliente} \blacktriangleleft [\ \mathit{conversaCompra} \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\mathit{prod}).\texttt{preço}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}?(d)\ ]\ ] \\
\mid \\
\mathit{Vendedor} \blacktriangleleft [\ \mathit{PreçoBD} \mid \\
\qquad \mathit{conversaCompra} \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\mathit{prod}).\texttt{acedePreço}^{\uparrow}!(\mathit{prod}). \\
\qquad\qquad\qquad\qquad\quad \texttt{valorPreço}^{\uparrow}?(p).\texttt{preço}^{\downarrow}!(p). \\
\qquad\qquad\qquad\qquad\quad \textbf{join}\ \mathit{Distribuidor} \cdot \texttt{ServiçoEntregas} \Leftarrow \texttt{artigo}^{\downarrow}!(\mathit{prod})\ ]\ ]) \\
\mid \\
\mathit{Distribuidor} \blacktriangleleft [\ \textbf{def}\ \texttt{ServiçoEntregas} \Rightarrow \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})\ ]
\end{array}$$

Neste momento a mensagem `compra` pode ser trocada entre cliente e vendedor na conversação *conversaCompra*, levando o sistema a evoluir para a seguinte configuração:

$$\begin{array}{l}
(\nu \mathit{conversaCompra}) \\
(\mathit{Cliente} \blacktriangleleft [\ \mathit{conversaCompra} \blacktriangleleft [\ \texttt{preço}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}?(d)\ ]\ ] \\
\mid \\
\mathit{Vendedor} \blacktriangleleft [\ \mathit{PreçoBD} \mid \\
\qquad \mathit{conversaCompra} \blacktriangleleft [\ \texttt{acedePreço}^{\uparrow}!(\mathit{prod}). \\
\qquad\qquad\qquad\qquad\quad \texttt{valorPreço}^{\uparrow}?(p).\texttt{preço}^{\downarrow}!(p). \\
\qquad\qquad\qquad\qquad\quad \textbf{join}\ \mathit{Distribuidor} \cdot \texttt{ServiçoEntregas} \Leftarrow \texttt{artigo}^{\downarrow}!(\mathit{prod})\ ]\ ]) \\
\mid \\
\mathit{Distribuidor} \blacktriangleleft [\ \textbf{def}\ \texttt{ServiçoEntregas} \Rightarrow \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})\ ]
\end{array}$$

Depois, as mensagem `acedePreço` e `valorPreço` são trocadas entre o código do serviço e o processo *PreçoBD* na conversação $\mathit{Vendedor}$. Nesse momento o vendedor envia para o cliente a mensagem `preço` na conversação *conversaCompra*, levando o sistema à seguinte configuração:

$$\begin{array}{l}
(\nu \mathit{conversaCompra}) \\
(\mathit{Cliente} \blacktriangleleft [\ \mathit{conversaCompra} \blacktriangleleft [\ \texttt{detalhes}^{\downarrow}?(d)\ ]\ ] \\
\mid \\
\mathit{Vendedor} \blacktriangleleft [\ \mathit{PreçoBD} \mid \\
\qquad \mathit{conversaCompra} \blacktriangleleft [\ \textbf{join}\ \mathit{Distribuidor} \cdot \texttt{ServiçoEntregas} \Leftarrow \texttt{artigo}^{\downarrow}!(\mathit{prod})\ ]\ ]) \\
\mid \\
\mathit{Distribuidor} \blacktriangleleft [\ \textbf{def}\ \texttt{ServiçoEntregas} \Rightarrow \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})\ ]
\end{array}$$

É então que o vendedor convida o distribuidor a juntar-se à conversação que está a decorrer (*conversaCompra*). O `ServiçoEntregas` é invocado através do idioma **join**, dando acesso ao distribuidor à identidade da conversação, permitindo que este se junte à tarefa de serviço através do estabelecimento de um terceiro ponto de acesso à conversação *conversaCompra*, desta feita detido pelo distribuidor. Como resultado do **join** o sistema evolui então para:

$$
\begin{array}{l}
(\nu \mathit{conversaCompra}) \\
(\mathit{Cliente} \blacktriangleleft [\ \mathit{conversaCompra} \blacktriangleleft [\ \texttt{detalhes}^{\downarrow}?(d)\ ]\ ] \\
\mid \\
\mathit{Vendedor} \blacktriangleleft [\ \mathit{PreçoBD} \mid \\
\qquad\qquad \mathit{conversaCompra} \blacktriangleleft [\ \texttt{artigo}^{\downarrow}!(\mathit{prod})\ ]\ ] \\
\mid \\
\mathit{Distribuidor} \blacktriangleleft [\ \mathit{conversaCompra} \blacktriangleleft [\ \texttt{artigo}^{\downarrow}?(p).\texttt{detalhes}^{\downarrow}!(\mathit{info})\ ]\ ]\ )
\end{array}
$$

Depois, as restantes trocas de mensagens podem ter lugar na conversação *conversaCompra*, nomeadamente a troca da mensagem `artigo` entre vendedor e distribuidor e a troca da mensagem `detalhes` entre distribuidor e cliente, completando-se esta colaboração de serviço.

Apesar de bastante simples, este exemplo demonstra desde já um cenário onde um participante se junta de forma dinâmica a uma conversação que está a decorrer e onde um participante interage em várias conversações de forma sequencial. Este tipo de sistemas levanta problemas difíceis a técnicas de verificação que visam garantir, em tempo estático de compilação, que os múltiplos intervenientes numa conversação seguem protocolos bem definidos de interacção e que tais sistemas nunca incorrem em pontos de bloqueio. Na próxima secção introduzimos as técnicas por nós propostas que endereçam estes problemas de análise mesmo nestes cenários complexos.

3.7 Análise de Conversações

Nesta secção apresentamos as principais ideias que estão na base das técnicas de verificação que propomos [9, 10] tendo em vista a verificação de propriedades chave de colaborações orientadas a serviço, nomeadamente que dão resposta às perguntas: "será que todos os participantes numa conversação com múltiplos intervenientes seguem protocolos de interacção bem definidos?" e "estará o sistema, onde os participantes intervêm em várias destas conversações livre de pontos de bloqueio?". A nossa abordagem lida com cenários onde múltiplos participantes interagem numa conversação — mesmo quando alguns deles são chamados dinamicamente a participar — e também quando os participantes interagem em várias conversações de forma sequencial, inclusivamente quando o acesso a algumas dessas conversações foi adquirido de forma dinâmica. Estes cenários não só apresentam um desafio do ponto de vista técnico, estando fora do alcance de abordagens anteriores, como também se revelam importantes na prática pois podemos encontrá-los em aplicações orientadas a serviços reais.

Introduzimos duas técnicas distintas mas complementares: para disciplinar interacção entre múltiplas partes introduzimos tipos de conversação, uma nova e flexível estrutura de tipos, capaz de uniformemente descrever as interacções internas e os interfaces externos dos sistemas, referidos como coreografias e contractos na terminologia dos serviços Web. Para garantir ausência de pontos de bloqueio, introduzimos um sistema de prova de progresso que se baseia numa noção de ordenação dos eventos do sistema e propagação destas ordenações nas comunicações. Descrevemos estes formalismos nas próximas secções.

3.7.1 Análise de Fidelidade à Conversação Baseada em Sistemas de Tipos

O ponto de partida para a análise das interacções entre as várias partes num sistema distribuído é ter uma forma como descrever os protocolos de interacção pretendidos. Tipicamente, por exemplo

em técnicas de análise que visam garantir propriedades de segurança, tais especificações dos protocolos consistem numa lista de troca de mensagens que indicam, para cada mensagem, quem é o emissor e quem é o receptor e que tipo de informação está contida na mensagem. Os trabalhos em tipos de sessão com múltiplos participantes, nomeadamente as abordagens de Honda et al. [20] e de Bettini et al. [6], também especificam o protocolo global das sessões de forma semelhante. Por exemplo, consideremos a especificação seguinte do protocolo de interacção entre cliente, vendedor e distribuidor do exemplo da secção anterior:

1. $\texttt{compra}(Ta)$: *Cliente* $\rightarrow$ *Vendedor*
2. $\texttt{preço}(Tp)$: *Vendedor* $\rightarrow$ *Cliente*
3. $\texttt{artigo}(Ta)$: *Vendedor* $\rightarrow$ *Distribuidor*
4. $\texttt{detalhes}(Td)$: *Vendedor* $\rightarrow$ *Cliente*

que descreve a interacção da compra como uma sequência de troca de mensagens que seguem uma ordem determinada e onde os intervenientes nas comunicações são identificados (por exemplo *Cliente* $\rightarrow$ *Vendedor* indica que a mensagem é enviada do *Cliente* para o *Vendedor*) e onde também são descritos os conteúdos das mensagens (por exemplo, Ta representa o tipo do conteúdo das mensagens `compra` e `artigo`).

Os tipos tradicionalmente aparecem associados a variáveis ou identificadores classificando o seu uso. Por exemplo, os tipos de dados inteiro ou "string" dizem-nos que uma variável deve ser usada para conter um valor inteiro ou uma sequência de caracteres, respectivamente. Porém, existem tipos que fornecem informação mais sofisticada, como por exemplo tipos comportamentais que especificam os protocolos usados nos canais de comunicação. Então podemos usar tipos comportamentais para verificar em tempo estático que os canais de comunicação são usados de forma correcta. A pergunta surge de como se verifica o correcto uso de um canal mediante um tipo.

Nas abordagens de Honda et al. [20] e de Bettini et al. [6] é explorada a ideia de projectar os tipos que caracterizam os protocolos globais nos papéis individuais dos participantes na sessão, permitindo então que o código de cada participante seja analisado mediante do seu próprio papel na colaboração. Contudo, estas abordagens não parecem adequadas para capturar colaborações que são estabelecidas dinamicamente, dado que os papéis individuais tem de ser à conhecidos à partida. Podemos então colocar a seguinte questão: será que se abstrairmos das identidades na especificação dos protocolos, podemos garantir que estes serão seguidos em tempo de execução? Na resposta a esta pergunta motivamos o nosso desenvolvimento, providenciando mais alguma intuição sobre os tipos de conversação antes de os definirmos formalmente.

Especificações de protocolos que não mencionam a identidade dos intervenientes nas comunicações parecem mais adequados para capturar sistemas onde as colaborações são estabelecidas dinamicamente, pois não restringem à partida a uma estrutura de comunicação rígida. Por um lado deixamos de poder certificar que as trocas de mensagens são feitas entre determinadas partes, mas, por outro lado, podemos na mesma verificar que as trocas de mensagens terão lugar entre quaisquer partes. Esta flexibilidade é um primeiro passo na direcção de suportar as configurações dinâmicas que pretendemos capturar. O tipo de conversação para a interacção da compra é então:

$$\tau\,\texttt{compra}(Ta).\tau\,\texttt{preço}(Tp).\tau\,\texttt{artigo}(Ta).\tau\,\texttt{detalhes}(Td)$$

onde é especificada uma sequência de troca de mensagens tal como anteriormente. Dada esta especificação do protocolo, somos confrontados com o problema de determinar como é que verificamos que a implementação respeita o protocolo definido. Para esse fim, podemos explorar a ideia de projectar as trocas de mensagens nas capacidades duais de comunicação que realizam a troca de mensagem — envio e recepção — que são as capacidades que de facto encontramos no código que implementa o papel de cada participante. Contudo, se projectarmos os protocolos globais directamente nos papéis individuais de cada participante acabamos novamente confinados a uma estrutura de comunicação estática.

Isto leva a um segundo e crucial passo no nosso desenvolvimento tendo em vista que pretendemos suportar configurações dinâmicas: a introdução de uma noção de projecção que é capaz de descrever as decomposições arbitrárias do protocolo no papel de uma ou mais partes. Para esse fim, os tipos de conversação misturam, ao mesmo nível na linguagem de tipos, especificações internas/"globais" (trocas de mensagens τ) e especificações de interface/locais (envio `!` e recepção `?`). Baseando-se nesta capacidade de misturar especificações locais e globais, introduzimos então uma relação ternária de fusão entre tipos, denotada $B = B_1 \bowtie B_2$, que explica como um comportamento B pode ser projectado em dois comportamentos compatíveis B_1 e B_2. Por exemplo, temos (entre outras) a projecção seguinte para o protocolo da compra:

$$\begin{aligned}&\tau\,\texttt{compra}(Ta).\tau\,\texttt{preço}(Tp).\tau\,\texttt{artigo}(Ta).\tau\,\texttt{detalhes}(Td)\\ &= !\,\texttt{compra}(Ta).?\,\texttt{preço}(Tp).?\,\texttt{detalhes}(Td)\\ &\quad \bowtie ?\,\texttt{compra}(Ta).!\,\texttt{preço}(Tp).\tau\,\texttt{artigo}(Ta).!\,\texttt{detalhes}(Td)\end{aligned}$$

que mostra a decomposição do protocolo global em dois tipos. O primeiro destes tipo caracteriza o papel do cliente na conversação de compra: primeiro envia a mensagem `compra`, depois recebe as mensagens `preço` e `detalhes`. O segundo tipo, que mistura tipos de interface e internos, caracteriza o papel combinado do vendedor e do distribuidor na conversação: as capacidades nas mensagens que são trocadas com o cliente são as duais em relação às do cliente, sendo também especificada a troca de uma mensagem internamente ao sistema vendedor-distribuidor (a mensagem `artigo` está anotada com τ). Este tipo caracteriza o subsistema composto por vendedor e distribuidor e pode ainda ser decomposto da seguinte forma:

$$\begin{aligned}&?\,\texttt{compra}(Ta).!\,\texttt{preço}(Tp).\tau\,\texttt{artigo}(Ta).!\,\texttt{detalhes}(Td)\\ &= ?\,\texttt{compra}(Ta).!\,\texttt{preço}(Tp).!\,\texttt{artigo}(Ta)\\ &\quad \bowtie ?\,\texttt{artigo}(Ta).!\,\texttt{detalhes}(Td)\end{aligned}$$

onde a decomposição gera os tipos que caracterizam os papéis individuais do vendedor e do distribuidor na conversação de compra. Por construção, se voltarmos a fundir todos estes fragmentos, voltamos a obter o protocolo global inicial. Estes vários tipos locais podem ser combinados pelo nosso sistema de tipos de uma forma composicional, permitindo a progressiva combinação comportamental das contribuições individuais de cada parceiro, até se chegar ao protocolo global. Mais, permitimos que uma parte do sistema seja tipificada inicialmente com uma parte arbitrária do protocolo que poderá depois ser delegada (parcialmente) de forma dinâmica, algo que é crucial para suportarmos a junção dinâmica de participantes a uma conversação. Por exemplo, isto permite-nos tipificar o tipo do serviço que implementa o papel do vendedor na conversação compra (`ServiçoCompras`) com o tipo do subsistema vendedor-distribuidor, desta forma suportando a delegação dinâmica que o vendedor faz de um fragmento do protocolo no momento do **join**.

Os tipos que caracterizam os fragmentos de conversação que são delegados nas comunicações são obtidos por decomposições (via a relação de fusão) do tipo detido pelo participante que executa a delegação. Por exemplo, quando analisamos o código do `ServiçoCompras`, no momento em que o **join** — onde se pede ao distribuidor que se junte à conversação — é tipificado, o tipo de conversação (residual) correspondente à participação do vendedor é $\tau\,\texttt{artigo}(Ta).!\,\texttt{detalhes}(Td)$. Nesse momento a extrusão do nome da conversação para o `ServiçoEntregas` vai ocorrer, para permitir que o distribuidor se junte à conversação. De referir que a conversação global vai ser respeitada em qualquer dos modos, visto que o fragmento da conversação delegado para o distribuidor é tipificado com $?\,\texttt{artigo}(Ta).!\,\texttt{detalhes}(Td)$ enquanto que o fragmento da conversação retido por vendedor é tipificado com $!\,\texttt{artigo}(Ta)$.

O sistema de tipos de conversações verifica, mesmo quando os protocolos são decompostos em fragmentos que podem ser dinamicamente delegados para outras partes, que o protocolo global é sempre respeitado. Dado que os tipos de conversação abstraem das identidades dos participantes,

$$
\begin{array}{lcll}
B & ::= & M.B \mid B_1 \mid B_2 \mid \mathbf{0} \mid \mathtt{rec}\,\mathcal{X}.B \mid \mathcal{X} & \text{(Comportamental)} \\
M & ::= & \mathsf{p}\, l(C) & \text{(Mensagem)} \\
\mathsf{p} & ::= & \mathtt{!} \mid \mathtt{?} \mid \tau & \text{(Polaridade)} \\
C & ::= & [B] & \text{(Conversação)} \\
L & ::= & n : C \mid L_1 \mid L_2 \mid \mathbf{0} & \text{(Localizado)} \\
T & ::= & L \mid B & \text{(Processo)}
\end{array}
$$

Figura 7: Sintaxe dos Tipos de Conversação.

o protocolo global pode ser decomposto de várias formas, permitindo por isso que existam várias implementações diferentes dos papéis na conversação. É mesmo possível tipificar sistemas com um número ilimitado de participantes, como é necessário para tipificar, por exemplo, um "broker" (mediador) de serviços.

O nosso sistema de tipos combina técnicas dos tipos lineares, comportamentais e de sessão (ver [19, 21, 24]). Tecnicamente, os ingredientes chave da nossa abordagem são a amálgama dos tipos globais e locais (no sentido de [20]) ao mesmo nível na linguagem de tipos e a definição da relação de fusão que garante, por construção, que os participantes tipificados com as projecções de um tipo podem ser compostos. A fusão compreende a noção de dualidade, no sentido que para cada B existem tipos $\overline{B}, B'$ tal que $B \bowtie \overline{B} = \tau(B')$, donde as sessões, que se baseiam na noção de dualidade, são casos particulares de conversações. Mais, a fusão de tipos permite maior flexibilidade na manipulação das projecções dos tipos de conversação, de uma forma aberta, como ilustrado no exemplo em cima. Em particular, a nossa abordagem permite que fragmentos de um tipo de conversação (uma coreografia) sejam distribuídos de forma dinâmica entre os participantes, enquanto se garante estaticamente que as interacções seguem os protocolos previstos.

Para fechar esta secção descrevemos o sistema de verificação formal que concretiza os conceitos acima descritos e enunciamos os principais resultados. Para não sobrecarregar a apresentação consideramos um modelo simplificado (nomeadamente sem direcções nas mensagens).

O sistema de tipos associa tipos de conversação a processo CC, onde os vários identificadores de conversações usados no processo são classificados com um protocolo. A sintaxe dos tipos de conversação é dada na Figura 7. Os tipos comportamentais incluem o prefixo de mensagem $M.B$, que representa processos que enviam, recebem ou trocam internamente a mensagem M e depois procedem como descrito em B — a polaridade da mensagem determina se o comportamento é um envio (!), recepção (?) ou troca interna (τ). Os tipos comportamentais incluem também a composição paralela $B_1 \mid B_2$, que representa processos que exibem dois comportamentos independentes B_1 e B_2 e captura comportamentos concorrentes, o tipo inacção $\mathbf{0}$ e o tipo recursivo $\mathtt{rec}\,\mathcal{X}.B$ que captura comportamentos infinitos. Os tipos mensagem $\mathsf{p}\, l(C)$ especificam, para além da polaridade p, a etiqueta da mensagem l e o tipo de conversação argumento C que caracteriza o fragmento da conversação delegado na comunicação.

Em geral um processo pode interagir em várias conversações. Os tipos localizados capturam os comportamentos dos processos nas várias conversações associando a cada identificador de conversação um tipo ($n : C$), permitindo-se a especificação de várias destas associações através da composição paralela ($L_1 \mid L_2$). Por fim, visto que os comportamentos dos processos não estão à partida localizados, um tipo processo $L \mid B$ permite combinar uma especificação localizada L com uma ainda não localizada B.

Por $P :: T$ denotamos a associação entre o processo P e o tipo T, que significa que o processo P interage nas conversações de acordo com os protocolos especificados em T. Dizemos que um processo P está bem tipificado se existe uma derivação usando as regras de tipificação cuja

$$\frac{P :: T_1 \quad Q :: T_2}{P \mid Q :: T_1 \bowtie T_2}(\textit{Par}) \qquad \frac{P :: T \mid a : [B] \quad (closed(B))}{(\boldsymbol{\nu} a)P :: T}(\textit{Res})$$

$$\frac{P :: L \mid B \mid x : C}{l?(x).P :: L \mid ?\, l(C).B}(\textit{Recepção}) \qquad \frac{P :: L \mid B}{l!(n).P :: (L \bowtie n : C) \mid !\, l(C).B}(\textit{Envio})$$

$$\frac{P :: L \mid B}{n \blacktriangleleft [P] :: (L \bowtie n : [B])}(\textit{Acesso}) \qquad \frac{P :: L \mid B_1 \mid x : [B_2]}{\textbf{this}(x).P :: L \mid (B_1 \bowtie B_2)}(\textit{Percepção})$$

Figura 8: Regras de Tipificação (Selecção).

conclusão seja $P :: T$. A Figura 8 apresenta algumas regras que associam tipos de conversação a sistemas CC. A regra (*Par*) caracteriza a composição paralela com o tipo obtido pela fusão dos tipos dos ramos paralelos. A fusão $\bowtie$ explica a composição de dois processos através da sincronização dos traços comportamentais. A regra (*Res*) expressa que a restrição de nomes está bem tipificada se o tipo da conversação restrita é um tipo fechado ($closed$), isto é, sem dependências para o exterior. A regra (*Recepção*) tipifica o respectivo prefixo, descrevendo o seu comportamento com o prefixo mensagem, onde o tipo argumento é o tipo da conversação identificada pela variável x. Por seu lado, na regra (*Envio*) fundimos o tipo delegado na mensagem, associando-o à conversação emitida na mensagem n, com o restante tipo localizado, por forma a contabilizar o comportamento delegado na comunicação. A regra (*Acesso*) especifica a fusão do tipo não localizado, associando-o à respectiva conversação, com o restante tipo localizado. Por sua vez, na regra (*Percepção*) fundimos o tipo da variável x com o tipo não localizado, pois x refere-se à conversação corrente.

Podemos agora enunciar os principais resultados. Em primeiro lugar provamos que processos bem tipificados reduzem sempre para processos bem tipificados. Visto que uma redução no processo pode implicar uma modificação nos tipos, definimos a redução nos tipos que essencialmente corresponde ao fecho reflexivo e por contexto da regra $\tau\, l(C).B \rightarrow B$.

Teorema 3.4 *Seja P um processo tal que $P :: T$. Se $P \rightarrow Q$ então existe $T \rightarrow T'$ tal que $Q :: T'$.*

O Teorema 3.4 garante-nos que cada redução no processo é explicada por uma redução no tipo.

Em segundo lugar garantimos que os processos bem tipificados não contêm erros, nomeadamente não contêm subprocessos que exibem o mesmo comportamento — uma corrida.

Proposição 3.5 *Seja P um processo bem tipificado. Então P está livre de erros.*

O Teorema 3.4 e a Proposição 3.5 garantem-nos directamente que um processo bem tipificado nunca incorre em erros durante a sua execução. Usamos $P \xrightarrow{*} Q$ para dizer que P tem uma sequência de reduções que levam até Q (fecho transitivo da redução).

Corolário 3.6 *Seja P um processo bem tipificado. Se $P \xrightarrow{*} Q$ então Q está livre de erros.*

O Corolário 3.6 garante que, numa qualquer sequência de reduções que origina num processo bem tipificado, existe sempre um único par de processos que podem trocar uma mensagem. Outra consequência do Teorema 3.4 é que qualquer redução no processo é explicada pela redução de um tipo mensagem de polaridade τ, implicando assim a propriedade fidelidade à conversação: todas as conversações seguem os protocolos prescritos pelos tipos.

Corolário 3.7 *Seja P um processo tal que $P :: T$. Então todas as conversações de P seguem os protocolos prescritos por T.*

3.7.2 Análise de Progresso em Conversações

Motivamos o nosso desenvolvimento com um exemplo. A seguinte especificação:

$$Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]$$

representa uma aplicação que está a tentar vender um artigo na $Amazon$ e depois usa o $eBay$ para o refornecimento do artigo em questão. O código especifica que na conversação $Amazon$ uma mensagem `compra` é recebida indicando o produto a ser vendido e depois uma mensagem `preço` é enviada indicando o valor da venda. Depois, a conversação $eBay$ é acedida e a mensagem `compra` é enviada para um vendedor especificando o artigo cujo "stock" é para ser reposto e uma mensagem `preço` é recebida contendo o valor da compra. Vamos supor que esta aplicação está a correr num contexto onde existe uma outra aplicação a correr o seguinte código:

$$eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]$$

A funcionalidade desta última é exactamente a mesma, a diferença é que está a trabalhar na outra direcção: vende no $eBay$ e refornece na $Amazon$. Quando consideramos o sistema resultante da composição destes dois processos em paralelo obtemos:

$$\begin{array}{l} Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ] \\ \mid\ eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ] \end{array}$$

onde podemos observar que o sistema está bloqueado pois ambos os processos estão à espera de receber uma mensagem. Contudo, os protocolos de conversação são seguidos, o que pode ser capturado pelo facto de ambas as conversações terem o tipo $\tau\,\texttt{compra}^{\downarrow}(Ta).\tau\,\texttt{preço}^{\downarrow}(Tp)$, mas os processos interagem nas conversações na ordem inversa. Neste exemplo o problema pode ser detectado por uma análise a olho nu, mas se várias linhas de código separassem as trocas de mensagens talvez mesmo uma simples dependência como esta poderia passar despercebida. Técnicas de análise formais podem ajudar a providenciar respostas a estes problemas de análise.

Tradicionalmente, abordagens que endereçam este tipo de problemas determinam se os eventos do sistema podem ser ordenados de uma forma bem fundada. Lynch [26] introduz uma das primeiras de tais abordagens, formalizando uma noção de ordenação para prevenir pontos de bloqueio em sistemas que acedem a recursos partilhados. Abordagens que endereçam sistemas especificados em Cálculo-π foram introduzidas mais recentemente, nomeadamente os trabalhos de Kobayashi [23] e, para tipos de sessão em particular, os trabalhos de Dezani-Ciancaglini et al. [14] e de Bettini et al. [6].

Voltemos ao exemplo "bloqueado" para ver como uma análise baseada na ordenação dos eventos pode identificar o problema. Os eventos no nosso caso são trocas de mensagens em conversações e são identificados pelo nome da conversação e pela etiqueta da mensagem. Para o código:

$$Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]$$

temos que a ordenação de eventos subjacente é tal que o evento $Amazon.\texttt{compra}$ é menor que o evento $Amazon.\texttt{preço}$ e por aí em diante, algo que denotamos por:

$$Amazon.\texttt{compra} \prec Amazon.\texttt{preço} \prec eBay.\texttt{compra} \prec eBay.\texttt{preço}$$

Contrariamente, para o código:

$$eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]$$

a ordenação de eventos é tal que:

$$eBay.\texttt{compra} \prec eBay.\texttt{preço} \prec Amazon.\texttt{compra} \prec Amazon.\texttt{preço}$$

Donde, para satisfazer os requisitos de ambos os processos, uma ordenação teria de ser tal que:

$$Amazon.\texttt{compra} \prec \ldots \prec eBay.\texttt{compra} \prec \ldots \prec Amazon.\texttt{compra} \prec \ldots$$

que não é uma ordem bem fundada. De facto não existe nenhuma ordem bem fundada para a composição destes dois processos. Podemos reproduzir análises semelhantes usando as abordagens de [14, 6]. Contudo, estes trabalhos têm algumas limitações no que toca à análise de processos onde os meios de comunicação (sujeitos à ordenação) são passados nas comunicações. Apresentamos agora como abordamos esta questão, usando para tal a seguinte variante do exemplo acima:

$$\begin{array}{l} \textit{eBayRevendedor} \blacktriangleleft [\\ \quad \texttt{venderEm}^{\downarrow}?(x). \\ \quad\quad x \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).eBay \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]\ ] \end{array}$$

que especifica um processo que faz a revenda de artigos disponíveis no $eBay$, desta feita fazendo esta revenda numa conversação cuja identidade é conhecida dinamicamente (através da mensagem `venderEm`). Consideremos também o seguinte código de um processo revendedor da $Amazon$:

$$\begin{array}{l} \textit{AmazonRevendedor} \blacktriangleleft [\\ \quad \texttt{venderEm}^{\downarrow}?(x). \\ \quad\quad x \blacktriangleleft [\ \texttt{compra}^{\downarrow}?(\textit{artigo}).\texttt{preço}^{\downarrow}!(p).Amazon \blacktriangleleft [\ \texttt{compra}^{\downarrow}!(\textit{artigo}).\texttt{preço}^{\downarrow}?(p)\ ]\ ]\ ] \end{array}$$

Apenas olhando para o código fonte o ponto de bloqueio não é evidente, pois depende de quais são as conversações em que os revendedores efectuam a revenda, nomeadamente se estes processos forem colocados em paralelo com o seguinte processo:

$$\textit{eBayRevendedor} \blacktriangleleft [\ \texttt{venderEm}^{\downarrow}!(Amazon)\ ] \mid \textit{AmazonRevendedor} \blacktriangleleft [\ \texttt{venderEm}^{\downarrow}!(eBay)\ ]$$

então este sistema irá incorrer num estado bloqueado semelhante ao que vimos antes.

O problema que se verifica neste exemplo só pode ser detectado se de alguma forma analisarmos como são ordenadas as referências para conversações que são passadas nas mensagens. A ordenação de eventos para o *eBayRevendedor* deve ser tal que:

$$x.\texttt{compra} \prec x.\texttt{preço} \prec eBay.\texttt{compra} \prec eBay.\texttt{preço}$$

donde qualquer instanciação de x tem de respeitar esta ordenação. De forma semelhante podemos verificar que para o processo $AmazonRevendedor$ temos uma ordenação prescrita similar, nomeadamente: $x.\texttt{compra} \prec x.\texttt{preço} \prec Amazon.\texttt{compra} \prec Amazon.\texttt{preço}$. Então, anexamos as ordens prescritas aos eventos onde as referências a conversações são passadas, por exemplo:

$$eBayRevendedor.\texttt{venderEm}.(x)(x.\texttt{compra} \prec x.\texttt{preço} \prec eBay.\texttt{compra} \prec eBay.\texttt{preço})$$

o que nos permite verificar que o nome que é enviado nestas comunicações cumpre (ou não) a ordem esperada pelo processo que recebe o nome. Podemos então excluir o sistema acima com a nossa técnica, dado que não existe nenhuma ordem bem fundada para os eventos neste sistema: o nome $Amazon$ é enviado no evento $eBayRevendedor.\texttt{venderEm}$ onde uma conversação x é esperada tal que $x.\texttt{compra} \prec \ldots \prec eBay.\texttt{compra}$ e o nome $eBay$ é enviado no evento $Amazon.\texttt{venderEm}$ onde é esperada uma conversação x com a seguinte ordenação $x.\texttt{compra} \prec \ldots \prec Amazon.\texttt{compra}$.

A nossa técnica permite-nos endereçar cenários complexos onde referências a conversações são passadas dinamicamente e o seu acesso é feito de forma sequencial pelo meio de acessos a outras conversações, permitindo assim que participantes se juntem a conversações e acedam de forma sequencial à conversação recebida e a outras, configurações que estão fora do alcance de trabalhos anteriores. Esta capacidade é crucial para endereçar sistemas onde os participantes

acedem de forma sequencial à conversação recebida dinamicamente e a outras, por exemplo para aceder a recursos locais ou para chamar parceiros remotos para que se juntem à conversação.

Do ponto de vista do desenvolvimento formal, introduzimos um sistema de prova que caracteriza a ordenação de processos CC mediante uma relação (bem fundada) de ordenação de eventos. Dizemos que o processo P está bem ordenado de acordo com a ordenação de eventos Γ, denotado por $\Gamma \vdash_n P$, se existe uma derivação no sistema de prova cuja conclusão seja $\Gamma \vdash_n P$, onde n é o nome da conversação corrente. Mostramos apenas a regra para o prefixo de envio:

$$\frac{(n.l.(x)\Gamma' \bot \Gamma) \vdash_n P \quad \Gamma'\{x/m\} \subseteq (n.l.(x)\Gamma' \bot \Gamma)}{\Gamma \vdash_n l!(m).P}(\textit{Envio})$$

A regra (*Envio*) especifica que o prefixo de envio está bem ordenado se a continuação do prefixo participa apenas em eventos de maior "ranking" em relação ao evento associado ao prefixo $(n.l.(x)\Gamma')$. Isto é assegurado pelo facto da continuação estar bem ordenada mediante $n.l.(x)\Gamma' \bot \Gamma$ que é a ordenação obtida seleccionando apenas eventos maiores que $n.l.(x)\Gamma'$ em Γ. Também se garante que a ordenação prescrita no evento associado ao prefixo $((x)\Gamma')$ é concordante com a ordenação global, quando instanciada com o nome da conversação emitido (m). Assim, garantimos que após a comunicação do nome na mensagem a ordenação global continua a ser respeitada.

Por fim, enunciamos os principais resultados. Primeiro provamos que um processo bem ordenado reduz sempre para um processo bem ordenado.

Teorema 3.8 *Seja P um processo tal que $\Gamma \vdash_n P$. Se $P \rightarrow Q$ então $\Gamma \vdash_n Q$.*

Depois provamos que um processo bem ordenado e bem tipificado (com um tipo fechado, isto é, que não tem dependências externas) onde ainda existam envios de mensagens pendentes tem sempre uma redução possível.

Teorema 3.9 *Seja P um processo tal que $\Gamma \vdash_n P$ e $P :: T$, onde $closed(T)$. Se em P existem envios de mensagens pendentes então $P \rightarrow Q$.*

O Teorema 3.9 garante que processos bem tipificados e bem ordenados nunca bloqueiam por haver um envio sem uma recepção correspondente. Esta propriedade garante que os serviços estão sempre disponíveis quando invocados e os protocolos onde são acedidas várias conversações de forma sequencial não incorrem em pontos de bloqueio.

4 Conclusões

Para finalizar este documento enumeramos as contribuições do trabalho aqui apresentado e apontamos algumas direcções de trabalho futuro. Em linhas gerais, apresentámos o estudo detalhado de um modelo de especificação de sistemas concorrentes e interactivos, particularmente adequado para capturar sistemas orientados a serviços. Usando este modelo, introduzimos novas técnicas formais que fornecem respostas automáticas a problemas de análise que endereçam propriedades cruciais, nomeadamente *fidelidade à conversação* e *progresso na conversação*, mesmo quando as conversações são estabelecidas dinamicamente e os participantes acedem de forma sequencial a várias conversações, cenários reais que estão fora do alcance de abordagens anteriores.

As principais contribuições do trabalho descrito neste documento são então:

- Em primeiro lugar, introduzimos o Cálculo das Conversações que define um mecanismo de comunicação baseado em trocas de mensagens etiquetadas relativas a contextos de conversação, e estudamos a sua teoria comportamental básica. Mostramos que esta pequena

linguagem permite desde já codificar abstracções de mais alto nível, nomeadamente a instanciação de serviços, a definição de serviços e o juntar de participantes de forma dinâmica a conversações que estão a decorrer; verificamos também que estas codificações admitem os tipos (derivados mecanicamente) esperados para as construções de mais alto nível.

- Em segundo lugar, definimos e formalizamos a noção de tipo de conversação. Os tipos de conversação são uma poderosa generalização dos tipos de sessão para conversações entre múltiplos participantes, que suportam a amálgama de especificações globais e locais ao mesmo nível na linguagem de tipos de uma forma legível e que permitem analisar sistemas onde fragmentos de conversações são delegados de forma dinâmica.

- Em terceiro lugar, propomos um sistema de tipos que associa tipos de conversação a sistemas orientados a serviços modelados em CC. Foi rigorosamente demonstrado que os processos validados pelas nossas regras de tipificação estão livres de erros de comunicação e corridas ("race conditions"), o que implica que sistemas bem tipificados gozam da propriedade de fidelidade à conversação.

- Por fim, apresentamos técnicas, igualmente demonstradas correctas, para garantir o progresso na conversação (ausência de bloqueios) de sistemas onde existam várias conversações a decorrer simultaneamente, eventualmente sendo acedidas de forma sequencial, explorando a combinação dos nomes das conversações e das etiquetas das mensagens para ordenar os eventos e, crucialmente, a propagação de ordenações nas comunicações, permitindo-nos resolver um problema até agora em aberto.

O trabalho descrito neste documento representa um passo importante na obtenção de um modelo de especificação geral e de técnicas de análise para endereçar interacção entre múltiplos participantes em sistemas distribuídos. Claramente, existem muitos pontos onde o trabalho pode ser melhorado e estendido, começando pelo facto que deixámos de fora aspectos como segurança, tolerância a falhas e mecanismos de interoperação, que interessa aprofundar.

Uma das direcções que parecem motivar trabalho futuro é o uso lógicas de especificação para capturar o comportamento de sistemas CC, tendo em vista o uso de um verificador de modelos ("model-checker") para verificar propriedades dos sistemas. Numa primeira abordagem, desenvolvemos uma ferramenta protótipo que permite verificar que um sistema CC segue uma determinada coreografia especificada em WS-CDL, recorrendo à tradução dos sistemas CC e das coreografias WS-CDL em especificações do Cálculo-π e em fórmulas de uma lógica temporal-espacial, respectivamente, que são alimentadas a um "model-checker" desenvolvido anteriormente.

Por fim, os nossos desenvolvimentos só atingem a sua máxima importância se forem transportados para a prática. Provámos a decidibilidade dos algoritmos de verificação, se os nomes ligados forem anotados com os tipos. Assim, iremos desenvolver uma linguagem de programação baseada no Cálculo das Conversações (extendendo uma linguagem "mainstream", e.g., java ou C#) e implementar ferramentas que façam a verificação dos tipos de conversação e da ordenação de eventos que sirvam de "proof-of-concept" da aplicabilidade nas nossas ideias. Começámos já a trabalhar neste sentido, tendo sido produzidos os primeiros protótipos.

Bibliografia

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.

[2] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, et al. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS Standard, 2007.

[3] J. Beatty, G. Kakivaya, D. Kemp, T. Kuehnel, B. Lovering, et al. Web Services Dynamic Discovery. Technical report, Microsoft Corporation and Co-Developers, 2005.

[4] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. Dovey, et al. Universal Description, Discovery, and Integration Version 3.0.2. Technical report, OASIS Committee Draft, 2005.

[5] Benjamin C. Pierce and D. Turner. Pict: a Programming Language Based on the π-Calculus. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[6] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR 2008, 19th International Conference on Concurrency Theory, Proceedings*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[7] E. Bonelli and A. Compagnoni. Multipoint Session Types for a Distributed Calculus. In *TGC 2007, Third International Symposium on Trustworthy Global Computing, Revised Selected Papers*, volume 4912 of *LNCS*, pages 240–256. Springer, 2008.

[8] M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and Pipelines for Structured Service Programming. In *FMOODS 2008, 10th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems, Proceedings*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.

[9] L. Caires and H. Vieira. Conversation Types. In G. Castagna, editor, *ESOP 2009, 18th European Symposium on Programming, Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 285–300. Springer-Verlag, 2009.

[10] L. Caires and H. Vieira. Conversation Types. *Theoretical Computer Science*, 2010. To appear.

[11] M. Carbone, K. Honda, and N. Yoshida. Structured Communication-Centred Programming for Web Services. In *ESOP 2007, 16th European Symposium on Programming, Proceedings*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.

[12] S. Chaki, S. Rajamani, and J. Rehof. Types as Models: Model Checking Message-Passing Programs. In *POPL 2002, 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Proceedings*, pages 45–57. ACM Press, 2002.

[13] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language Version 2.0. Technical report, W3C Recommendation, 2007.

[14] M. Dezani-Ciancaglini, U. de'Liguoro, and N. Yoshida. On Progress for Structured Communications. In *TGC 2007, Third International Symposium on Trustworthy Global Computing, Revised Selected Papers*, volume 4912 of *LNCS*, pages 257–275. Springer, 2008.

[15] M. Dezani-Ciancaglini, S. Drossopoulou, D. Mostrous, and N. Yoshida. Objects and Session Types. *Information and Computation*, 207(5):595–641, 2009.

[16] M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. Hunt, et al. Language Support for Fast and Reliable Message-Based Communication in Singularity OS. In *Proceedings of the EuroSys 2006 Conference*, pages 177–190. ACM Press, 2006.

[17] C. Fournet, F. Le Fessant, L. Maranget, and A. Schmitt. JoCaml: A Language for Concurrent Distributed and Mobile Programming. In *AFP 2002, 4th International School on Advanced Functional Programming, Revised Lectures*, volume 2638 of *LNCS*, pages 129–158. Springer, 2003.

[18] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. Nielsen, et al. Simple Object Access Protocol Version 1.2. Technical report, W3C Recommendation, 2007.

[19] K. Honda, V. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP 1998, 7th European Symposium on Programming, Proceedings*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.

[20] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL 2008, 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Proceedings*, pages 273–284. ACM Press, 2008.

[21] A. Igarashi and N. Kobayashi. A Generic Type System for the π-Calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.

[22] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language Version 1.0. Technical report, W3C Candidate Recommendation, 2005.

[23] N. Kobayashi. A New Type System for Deadlock-Free Processes. In *CONCUR 2006, 17th International Conference on Concurrency Theory, Proceedings*, volume 4137 of *LNCS*, pages 233–247. Springer, 2006.

[24] N. Kobayashi, B. Pierce, and D. Turner. Linearity and the π-Calculus. In *POPL 1996, 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Proceedings*, pages 358–371. ACM Press, 1996.

[25] N. Kobayashi, K. Suenaga, and L. Wischik. Resource Usage Analysis for the π-Calculus. *Logical Methods in Computer Science*, 2(3):1–42, 2006.

[26] N. Lynch. Fast Allocation of Nearby Resources in a Distributed System. In *STOC 1980, Twelfth Annual ACM Symposium on Theory of Computing, Proceedings*, pages 70–81. ACM Press, 1980.

[27] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[28] R. Milner. The Polyadic π-Calculus: A Tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer, 1993.

[29] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[30] B. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[31] V. Vasconcelos, S. Gay, and A. Ravara. Type Checking a Multithreaded Functional Language with Session Types. *Theoretical Computer Science*, 368(1-2):64–87, 2006.

[32] H. Vieira, L. Caires, and J. Seco. The Conversation Calculus: A Model of Service-Oriented Computation. In S. Drossopoulou, editor, *ESOP 2008, 17th European Symposium on Programming, Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2008.