Thomas Pedro Noronha

Licenciado em Ciências da Engenharia Eletrotécnica

e de Computadores

Geração automática de sistemas compatíveis com o Arrowhead Framework utilizando IOPT-Tools

Dissertação para a Obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Doutor Filipe de Carvalho Moutinho, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Doutor Rogério Alexandre Botelho Campos Rebelo, Investigador, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente: Prof. Doutor Luís Filipe Lourenço Bernardo

Arguente: Prof. Doutor João Almeida das Rosas

Vogal: Prof. Doutor Filipe de Carvalho Moutinho

Setembro, 2018

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Geração automática de sistemas compatíveis com o Arrowhead Framework utilizando IOPT-Tools

Dedicatória aos meus amigos, mãe, irmã e namorada…

Agradecimentos

Em primeiro lugar pretendendo agradecer ao Professor Doutor Filipe Moutinho e ao Professor Doutor Rogério Campos-Rebelo pela disponibilidade e profissionalismo, tal como a boa disposição, conselhos e orientação. Estou muito agradecido por me terem proposto este tema de dissertação, e ainda pela oportunidade de elaboração de um artigo.

Quero agradecer ao Professor Doutor Luís Gomes pela oportunidade em fazer parte do corpo de docente da cadeira de Sistemas Lógicos.

Quero agradecer a todos os docentes e colegas da instituição FCT da Universidade Nova de Lisboa, que contribuíram para o meu percurso académico.

Quero agradecer aos meus amigos e familiares que de alguma forma seja ela direta ou indireta, me ajudaram ao longo dos cinco anos de mestrado.

Um especial agradecimento aos meus amigos Pedro Rodrigues e Ricardo Mota que me acompanharam nestes últimos anos; à minha mãe, Fátima Pedro, pela paciência, oportunidade e privilégio de estudar; e por último, mas não menos importante, à minha namorada, Mariana Sousa, pelo amor, incentivo e por ter acreditado em mim.

Estou eternamente grato por todo o bem que me fizeram.

Resumo

Nesta dissertação propõe-se uma extensão às ferramentas IOPT-Tools por forma a suportarem a geração sistemas compatíveis com o *Arrowhead Framework*. Os sistemas gerados têm a capacidade de se registar (como fornecedores) ou procurar (enquanto consumidores) serviços no *Arrowhead Framework*. O *Arrowhead Framework* conta com um conjunto de ferramentas que permitem, o registo, a pesquisa e a orquestração de sistemas produtores e/ou consumidores de serviços, para que possam interagir dinamicamente, sem ser necessário que se conheçam previamente. O uso de IOPT-Tools permite tirar proveito das suas capacidades de edição e validação de modelos, bem como da geração automática de código.

Neste trabalho foi necessário instalar e realizar um estudo sobre as várias ferramentas do *Arrowhead Framework* e das IOPT-Tools, identificar alterações/extensões a efetuar às IOPT-Tools e desenvolver uma biblioteca em C capaz de suportar a interação dos sistemas com o A*rrowhead Framework*. A validação das alterações/extensões às IOPT-Tools, bem como da biblioteca desenvolvida foram feitas através da geração e execução de um conjunto de sistemas compatíveis com o *Arrowhead Framework*.

Palavras-chave: indústria 4.0; IoT; Arrowhead Framework; redes de Petri; IOPT-Tools; VPL;

Abstract

In this dissertation it is proposed an extension to the IOPT-Tools tools in order to support the generation of systems compatible with the Arrowhead Framework. The generated systems have the ability to register (as providers) or search (as consumers) for services in the Arrowhead Framework. *Arrowhead framework* has a set of tools that allow the registration, research and orchestration of producer and/or service consumer systems, so that they can interact dynamically, without having to know each other beforehand. The use of IOPT-Tools allows you to take advantage of your model editing and validation capabilities as well as automatic code generation.

In this work it was necessary to install and perform a study on the various tools of the *Arrowhead Framework* and IOPT-Tools, to identify changes/extensions to be made at IOPT-Tools and develop a C library capable of supporting the interaction of the systems with the Arrowhead Framework. The validation of the changes/extensions to IOPT-Tools as well as the developed library were done through the generation and execution of a set of systems compatible with the Arrowhead framework.

Keywords: industry 4.0; IoT; Arrowhead Framework; Petri nets; IOPT-Tools; VPL;

Índice Geral

Índice de Tabelas

Índice de Figuras

Lista de acrónimos

CoAP - *Constrained Application Protocol*

DS-Pnets - *Data-flow, Signals and Petri nets*

HTTP - Hypertext Transport Protocol

IL - *Instruction List*

IOPT - *Input/Output Place transition*

IoT - Internet das coisas

MQTT - *Message Queuing Telemetry Transport*

OPC UA - *Open Platform Communications Unified Architecture*

PLC - Programmable logic controller

SOA - Arquitetura orientada a serviços

VHDL - Very High Speed Integrated Circuits

VPL - *Visual Programming Languages*

XMPP - *Extensible Messaging and Presence Protocol*

Preâmbulo

O futuro é o presente, a 4ª revolução industrial [1] já não é um conceito abstrato dos empresários e políticos, a revolução digital está a acontecer agora e é mais poderosa, crescendo mais rápido do que pensamos. Esta mudança traz consigo, tanto iluminação como sombra. A modernização dos edifícios, cidades e indústria criando sistemas inteligentes, flexíveis e resilientes, é o resultado do esforço de milhares de sonhadores nos últimos 70 anos. As casas inteligentes tornam o nosso dia-a-dia mais confortável e prático, as cidades inteligentes cada vez mais ecológicas, seguras e limpas, e as fábricas cada vez mais rentáveis, eficientes e organizadas. Isto são tudo coisas boas e desejáveis, no entanto o efeito é controverso pois por um lado a tecnologia veio para beneficiar as pessoas, tirando cargos duros e repetitivos, mas por outro, tira empregos às pessoas.

O facto que a tecnologia está a substituir os trabalhadores seja nas fábricas, como nos bancos, caixas, e em qualquer outra área de negócio, prova que estamos perante a uma mudança muito maior que uma simples nova revolução industrial, pois pela primeira vez na história, o humano já não é visto como um meio de produção. A indústria 4.0 veio para "matar" trabalhos, substituindo cargos repetitivos por máquinas eficientes e incansáveis, mas o objetivo não é tirar emprego às pessoas, é criar-lhes novos empregos mais criativos, menos repetitivos e mais gratificantes.

1 Introdução

Neste capítulo pretende-se contextualizar este trabalho de dissertação, apresentar a motivação que levou à realização deste trabalho, bem como, os seus objetivos e a estruturação utilizada na escrita deste documento.

1.1 Contextualização

As três revoluções industriais precedentes impuseram conceitos de maquinaria, depois produção em massa, e automação. No entanto, a atual foca-se sobretudo na otimização de processos com base na conectividade e interação entre dispositivos e utilizadores, visando a criação de novos produtos, serviços e mercados, otimização dos processos de produção, produção personalizada, redução de custos, entre outros (Figura 1.1) [2].

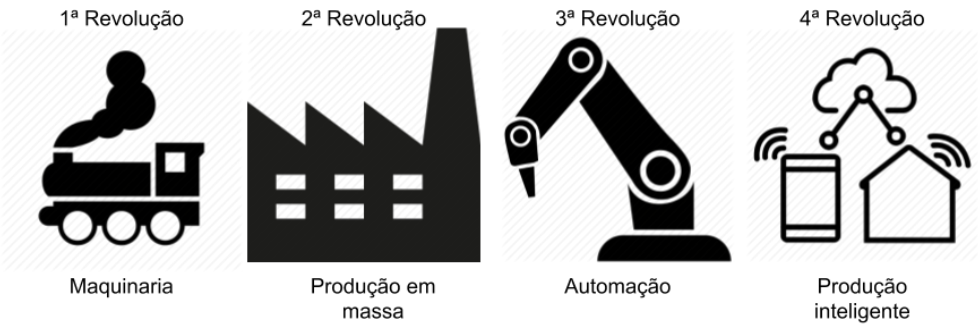

Figura 1.1 - Revoluções Industriais.

A chamada indústria 4.0, pelos alemães, ou manufatura avançada, pelos americanos e chineses, é um movimento aplicado a “fábricas inteligentes”, “cidades inteligentes” e “Sistemas inteligentes”, que surge por via da maturidade de um conjunto alargado de tecnologias tais como, realidade aumentada, *Cloud Computing*, captura e análise de dados, deteção de fraude e autenticação, impressão 3D, a Internet das Coisas, Manufatura aumentada, utilizando estes conceitos tecnológicos de modo a proporcionar um aproveitamento dos sistemas implementados na indústria e serviços ditos tradicionais [3][4].

Atualmente os componentes de uma fábrica produzem dados para tomar decisões autonomamente. No entanto, podem necessitar de informação externa para tomar decisões. Devido ao desafio relativamente à personalização dos produtos, é conveniente abordar um cenário onde o produto participa na produção, a fim de cada máquina saber quais as operações a efetuar ao produto, seja para direcionar o produto nas passadeiras ou na manufaturação.

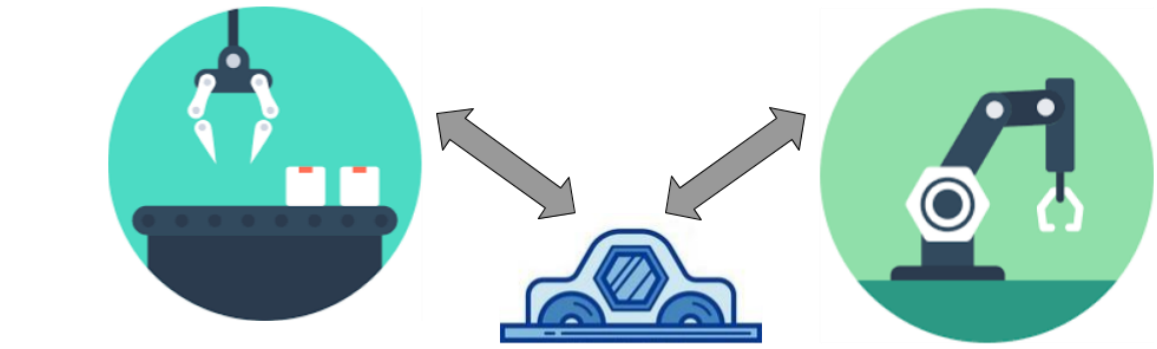

Figura 1.2 - Interação do produto com máquinas industriais.

A Internet das Coisas (IoT - *Internet of Things*, em inglês) [5] consiste na interligação remota de dispositivos físicos, máquinas, servidores, veículos, sistemas, utilizadores, entre outros, a fim de partilhar informação. Conectar “tudo” com “todos” permite criar sistemas com uma organização descentralizada que resulta em processos modulares mais resilientes, possibilitando a criação de novas soluções e novos serviços.

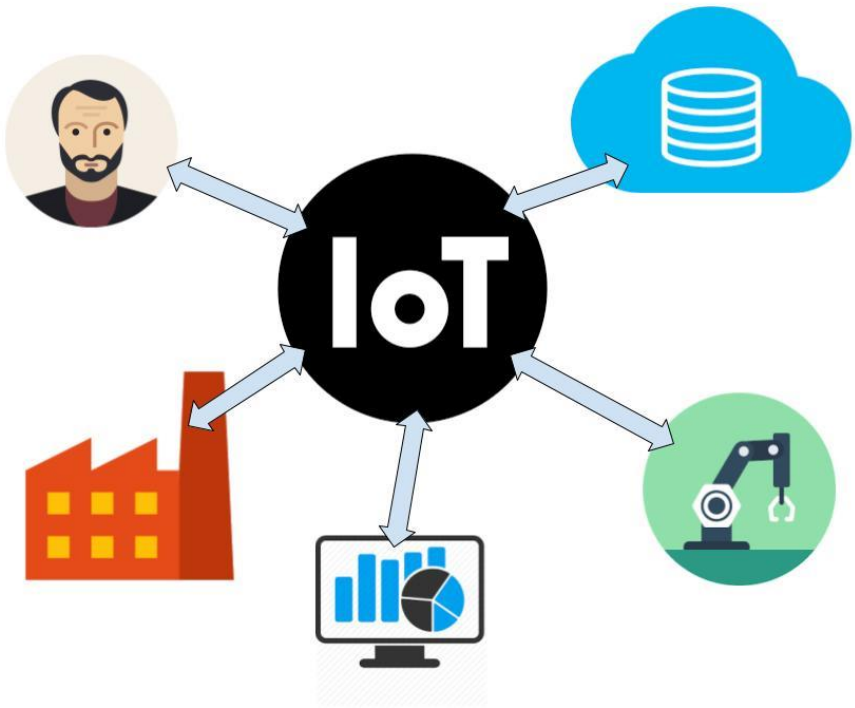

Figura 1.3 - Interconexão IoT.

O *Arrowhead Framework* tem vindo a ser desenvolvido no âmbito de dois projetos europeus: Arrowhead Project [6] (já terminado) e Productive 4.0 Project [7] (a decorrer). Productive 4.0 é um projeto de inovação europeu, que visa validar as potencialidades da indústria digital de modo a manter a liderança das indústrias europeias. Este projeto envolve diversas instituições europeias, todas focadas em integrar hardware e software IoT na indústria digital.

Manufatura, edifícios inteligentes, mobilidade elétrica, geração de energia e mercados virtuais de energia são os domínios de aplicação do projeto Productive 4.0. Para servir estes domínios, foram desenhadas arquiteturas baseadas em soluções SOA (*Service-Oriented Architecture*). O objetivo principal do *Arrowhead Framewrok* é a inter-conectividade entre diferentes sistemas, mantendo as vantagens dos sistemas SOA.

O *Arrowhead Framework* conta com um conjunto de ferramentas de software que permitem, entre outras funcionalidades, o registo e pesquisa de dispositivos/sistemas, para que seja possível estabelecer ligações dinâmicas entre dispositivos numa rede de modo a partilharem informação, sem ser necessário que os dispositivos se conheçam previamente. Estas ferramentas utilizam princípios de arquitetura orientada a serviços (SOA). Isto significa que um dispositivo que se ligue à rede pode comunicar com quaisquer dispositivos, desde que use os serviços das ferramentas do *Arrowhead Fram*ework. O *Arrowhead Framework* disponibiliza vários serviços, que permitem que os dispositivos/componentes se liguem a outros dispositivos que dispõem de

informação relevante, baseando-se numa estrutura distribuída onde cada dispositivo pode fornecer informação e outros podem consumir informação.

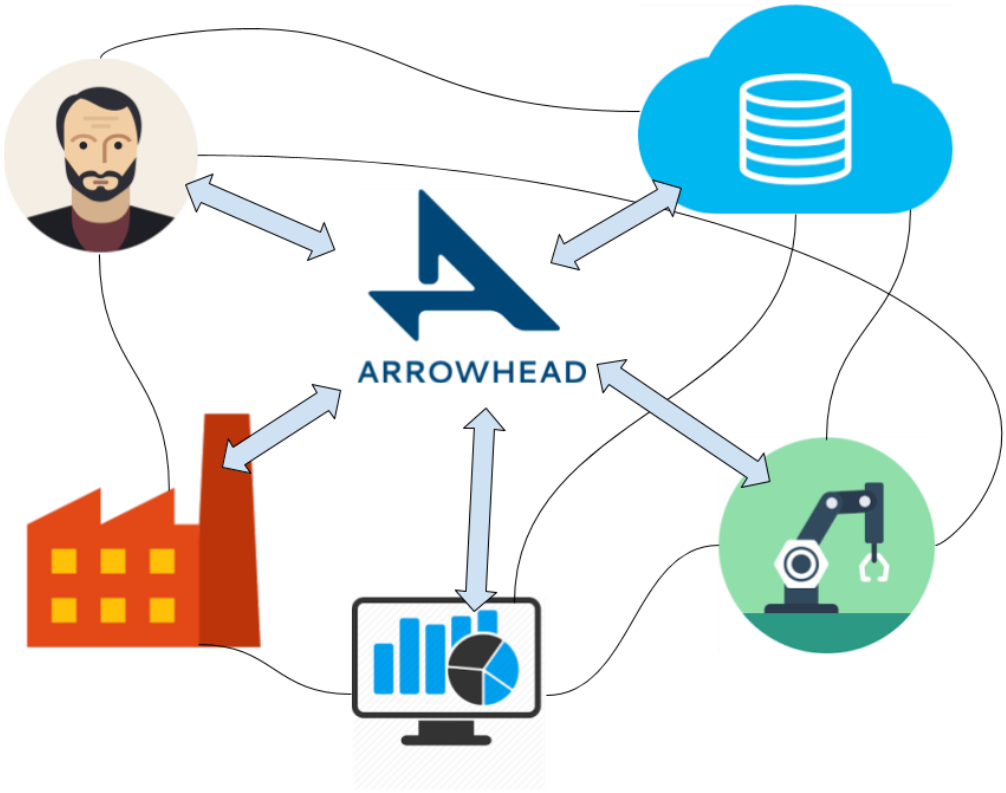

Figura 1.4 - *Arrowhead Framework* IoT dinâmico.

Devido ao aumento de complexidade dos sistemas de automação, verifica-se uma tendência na criação de ferramentas, que de alguma forma permitem poupar tempo e/ou recursos no desenvolvimento destes sistemas. VPL (Visual Programming Languages) [8] são ferramentas gráficas que geralmente são desenvolvidas especificamente para cada sector/área de operação. No entanto, também podem ser menos genéricas, focando-se num método de modelação particular. As IOPT-Tools [9], constituem uma ferramenta gráfica que permitem modelar e validar controladores com redes de Petri IOPT (*Input/Output Place Transition*), gerando código automaticamente. Do ponto de vista de desenvolvimento de automação industrial, as IOPT-Tools são adequadas para modelar sistemas de controlo, disponibilizando diversas funcionalidades, como por exemplo capacidade de simular modelos IOPT, gerar automaticamente código, monitorizar e controlar remotamente, etc…

É importante desenvolver mecanismos que permitem juntar conceitos de IoT com ferramentas existentes a fim de facilitar desenvolvimento de soluções e melhorar a qualidade dos processos. Assim, na secção seguinte, pretende-se apresentar os motivos que levaram à escolha deste projeto de dissertação.

1.2 Motivação

A FCT/UNL é atualmente parceira do projeto Productive 4.0, estando neste momento a contribuir para o desenvolvimento do projeto Arrowhead Framework, contextualizado na indústria 4.0. Em adição, a FCT/UNL é responsável pelo desenvolvimento das IOPT-Tools, que são ferramentas gráficas de modelação de controladores. Assim, pretende-se contribuir no desenvolvimento destas ferramentas, com o intuito em acrescentar funcionalidades com aplicação na indústria 4.0.

1.3 Objetivos

O objetivo principal deste trabalho é permitir que os controladores/sistemas desenvolvidos pelas IOPT-Tools sejam compatíveis com o Arrowhead Framework. Isto é, estender as ferramentas de modelação IOPT e gerador automática de código, de forma a que as IOPT-Tools permitam o desenvolvimento de modelos fornecedores de serviços (daqui para a frente chamados de *providers*) e/ou consumidores de serviços (daqui para a frente chamados de *consumers*). Os serviços gerados terão de ter a capacidade de se registar no *Arrowhead Framework* (*providers*) ou procurar *providers* no *Arrowhead Framework* (*consumers*). Por forma a atingir o objetivo principal, foi necessário:

- Fazer um estudo prévio das características e potencialidades do Arrowhead Framework, bem como das ferramentas IOPT e das redes de Petri IOPT;
- Identificar que alterações/extensões são necessárias efetuar às ferramentas IOPT, nomeadamente no editor e no gerador de código C;
- Alterar/estender as ferramentas IOPT;
- Desenvolver uma biblioteca em C capaz de suportar a interação com o Arrowhead Framework;
- Desenvolver controladores/sistemas compatíveis com o *Arrowhead Framework* utilizando as ferramentas IOPT alteradas/ estendidas;
- Instalar e executar as ferramentas base do Arrowhead Framework;
- Testar os controladores desenvolvidos com as ferramentas do Arrowhead Framework.

1.4 Estrutura

Esta dissertação foi dividida em oito capítulos: Introdução; Estado de Arte; Arrowhead Framework; Projeto biblioteca C; Extensão às IOPT-Tools; Teste e validação da contribuição; Conclusão e Trabalhos Futuros.

A introdução pretende clarificar o tema desta dissertação, tal como os problemas envolvidos, motivação e objetivos finais para o projeto.

No segundo capítulo fez-se um levantamento de características da indústria 4.0, problemas na indústria, faz-se um estudo das tecnologias modernas e tendências futuras com especial foco nas características de sistemas IoT, impacto destas tecnologias na indústria e consequentes benefícios. É também feito um levantamento de I&D relativamente aos sistemas IoT. Também se realiza um levantamento de ferramentas gráficas (VPL) para modelação de sistemas e controladores existentes na atualidade, com interesse em analisar e comparar formalismos de modelação tal como suas características inovadoras e suas limitações. Pretende-se também fazer uma descrição geral às IOPT-Nets e IOPT-Tools e suas funcionalidades. São introduzidas as IOPT-Flow e é feita uma comparação sumativa destas ferramentas com o Matlab/Simulink e Labview.

O capítulo três apresenta o *Arrowhead Framework* e as funcionalidades associadas à ferramenta, descrevendo os serviços, modo de funcionamento teórico e os principais sistemas (*Cores systems* - sistemas de serviços*)* do Arrowhead Framework.

No quarto capítulo apresenta-se o resultado do desenvolvimento da biblioteca C Arrowhead Framework.

No capítulo cinco são apresentadas algumas hipóteses propostas para a realização desta dissertação explicando com detalhe os passos necessários para estender as IOPT-Tools.

No capítulo seis pretende-se apresentar resultados e discutir os testes e validação da dissertação.

O capítulo sete é reservado à conclusão da dissertação e trabalhos futuros. É indicando resumidamente o que foi desenvolvido, quais os resultados esperados e quais os resultados efetivos. Por fim, enumera-se alguns trabalhos futuros associados à ferramenta desenvolvida.

2 Estado de Arte

2.1 Indústria 4.0

Com o avanço da tecnologia digital, foi possível construir software em cima de sistemas de controlo de forma a permitir um nível de abstração sobre sistemas. Esta camada de software, não só permite o desenvolvimento de sistemas de controlo mais flexíveis, mas também oferece um aproveitamento no uso dos dados adquiridos por sensores da *shop floor* e consequentemente aplicar melhores respostas nos atuadores, fazendo assim uma melhor gestão dos recursos, reduzindo a quantidade e gravidade de falhas [10].

Produtividade é essencial para descrever a indústria. Na história existiram 3 revoluções industriais, as revoluções criadas pelas máquinas a vapor, pela produção em massa e pela a introdução da automação. Estes momentos históricos ficaram marcados por enormes crescimentos na economia. No entanto, o crescimento económico foi uma consequência da produtividade [11]. O objetivo principal é a produtividade. A indústria identificou diversas características que favorece a produtividade, como por exemplo: A robotização, *Big Data*, simulação, internet das coisas, manufatura aditiva e realidade aumentada.

Atualmente, estas características estão cada vez mais presentes em ambiente de manufatura, mas também em cidades inteligentes, edifícios inteligentes, sistemas inteligentes, e empresas dinâmicas, o que criou uma onda de euforia sobre estas tecnologias. O que hoje chamamos de 4ª revolução industrial.

A domótica é uma área em desenvolvimento desde há várias décadas. No entanto, a 4ª revolução industrial trouxe novos conceitos para controlo de casas. Percebeu-se que estas tecnologias são bastante benéficas nos edifícios inteligentes. É interessante conectar a maior parte dos dispositivos de forma a fazer uma extração de dados para uma infraestrutura de nuvem, permitindo a monitorização do edifício, simulação, estudo estatístico, controlo remoto, etc… Existem diversos exemplos de aplicação destes conceitos, por exemplo cada sala pode ter sensores de

movimento, sensores de luminosidade, e com isto podermos "programar" o edifício a acender as luzes do escritório consoante a deteção de presença e ajustar a luminosidade, equilibrando com a luz natural. Este exemplo mostra que estes conceitos podem trazem melhorias na otimização de consumo ecológico nos edifícios.

2.1.1 Características da Indústria 4.0

A 4ª revolução industrial traz algumas mudanças à indústria 3.0. A simulação das fábricas permite organizar e dimensionar as linhas de produção. Por outro lado, os sensores distribuídos na *shop floor* permitem monitorizar os processos, analisar estatística de informação recolhida por forma atuar sobre falhas ou até mesmo prevenir avarias em tempo real. Outro princípio é a descentralização dos processos de produção, que em vez de ser realizada por um servidor centralizado, poderá ser realizada por um sistema descentralizado, de forma a tornar o sistema mais resiliente a falhas, disponibilizando informação mais detalhada sobre a função desempenhada. Outro aspeto é a aplicação de internet das coisas, que permite a conexão de diversos pontos na mesma rede, permitindo a reorganização tecnológica de produção e serviços, necessário para a obtenção de melhores resultados. Por fim, a modularização da produção de acordo com a requisição, acoplamento e desacoplamento de módulos de manufatura, permite manter o ambiente fabril flexível para reprogramar as tarefas das máquinas.

2.1.1.1 Big Data

Genericamente, *Big data* [12] é definida como uma grande acumulação de dados que contém informação relevante, mas que seja difícil de coletar, armazenar e analisar. A *Big Data* tem três principais características: Volume, velocidade e variedade. Nos dias de hoje tem havido um enorme crescimento de dados armazenados em infraestruturas digitais. Além disso, os dados foram ficando mais diversificados, mais complexos, menos estruturados. Adicionalmente, também tem aumentado a necessidade em processar essa informação rapidamente.

A *Big Data* tem como principal interesse disponibilizar serviços ou ferramentas de analise estatística de dados proveniente de sistemas externos fornecedores, que ajuda o cliente (ou consumidor) a entender essa informação e usa-la.

Big Data tem as seguintes características:

- Grande armazenamento de informação útil para o cliente.
- Variedade de dados, provenientes de diversos sistemas em rede colaborativa.
- Disponibilidade de serviços capaz de processar dados da infraestrutura em tempo real.

2.1.1.2 Realidade aumentada

A realidade aumentada consiste na extração de informação proveniente da realidade a fim de o utilizador obter melhor perceção visual do ambiente envolvente ou sistema. Esta tecnologia é aplicada em várias áreas científicas tais como: medicina, educação, arquitetura, engenharia, jo-

gos tecnológicos. Por exemplo, a criação de plantas de edifícios para registo arquitetónico. Capacetes inteligentes para bombeiros (C-Thru [13]) que permitem visualizar temperaturas e cancelamento de ruido para focar em pedidos de socorro. Controle e manutenção de equipamentos em indústrias. E também na área da publicidade e marketing, os clientes podem usar óculos VR para obterem uma visão mais realista do produto.

2.1.1.3 Simulação

A simulação de produção (com software como Arena [14] ou Flexsim Express [15]) na fase de planeamento e desenvolvimento das linhas de produção das fábricas, permite fazer uma analise de resultados do modelo, estimando custos de produção, qualidade de serviço/produto, testar modelos e a organização de produção, a fim de prevenir custos desnecessários, problemas de qualidade e falhas. Esta metodologia de planeamento é cada vez mais comum, sendo claramente uma mais-valia para a indústria 4.0, pois permite planear, projetar/dimensionar, testar produtos e realizar o controlo de qualidade virtualmente antes de iniciar produção.

2.1.1.4 Manufatura aditiva

O processo de transporte é responsável no aumento considerável do custo final do produto. Por isso, a manufatura aditiva permite que a produção seja realizada localmente mais perto do consumidor, eliminando uma percentagem significativa do custo de transporte.

Com a aplicação de *Internet of Things*, *Big, Data* e impressão 3D, estas tecnologias permitem criar um novo conceito de manufatura. Pretende-se desenvolver o produto e enviar o projeto para uma fábrica mais perto do consumidor. Manufatura aditiva e impressão 3D ainda é um conceito a ser desenvolvido. No entanto, no futuro as impressoras 3D vão ter capacidade de impressão flexível sem a necessidade de preparação. [16]

2.1.1.5 Personalização dos artigos

Já se verificam negócios que oferecerem ao consumidor a escolha de algumas características do produto, tal como cor, tamanho, extras, entre outros. No mercado automóvel existem modelos de marcas onde é oferecido ao cliente a opção de personalizar o automóvel de modo a criar um produto "único" criado especificamente para o cliente.

O cliente atual deseja comprar produtos específicos para ele e exige rapidez na entrega. Para isso, e para produzir esses produtos únicos em grande escala, é necessário olhar para cada produto como uma unidade, atribuindo-lhe uma identificação de modo a que possa ser identificado o seu processo de produção e agir sobre ele. O produto participa na produção, pois ele é identificado numa base de dados onde está registado o pedido do cliente, aplicando sobre o produto as ações necessárias. Este género de abordagem facilita a customização da produção, porque não depende diretamente da interação do ser humano, produzindo autonomamente em alta escala produtos diferentes.

Em certos mercados faz sentido o cliente participar diretamente na prototipagem do produto, de forma a ajudar o produtor a entender as necessidades do mercado, por exemplo no mercado desportivo, procura-se assistência dos profissionais desportivos de modo a desenvolver produtos indicados à prática desportiva.

2.1.1.6 Internet das coisas (IoT)

Internet das coisas, é um conceito para a ligação de dispositivos e pessoas em rede com a finalidade de extrair informação relevante, permitindo a criação de novos serviços ou otimizar aproveitamento de sistemas. A internet das coisas pretende:

- Conectar coisas
- Monitorar coisas
- Localizar coisas
- Controlar coisas

Estas características permitem não só estabelecer ligação com dispositivos e pessoas, mas também desenvolver sistemas de sistemas. Por exemplo camaras presentes em autoestradas podem monitorizar diversas variáveis, tais como tráfego, condições meteorológicas da estrada, etc... Por sua vez, estes sistemas podem partilhar informação com outros sistemas, sistemas de transportes, metro, aeroporto, centro metrológico, distribuição, lixeiras, etc…

Estima-se 50 biliões de dispositivos ligadas à internet até 2020.

O Google Trends é uma ferramenta do Google que permite a monitorização da popularidade de temas e assuntos de pesquisa num gráfico temporal e também em histograma geográfico. Com esta ferramenta podemos visualizar a frequência de pesquisa sobre o tema da internet das Coisas.

O eixo horizontal do gráfico representa tempo, e o eixo vertical corresponde há frequência que o tema é pesquisado globalmente, admitindo as equivalências linguísticas do assunto de pesquisa.

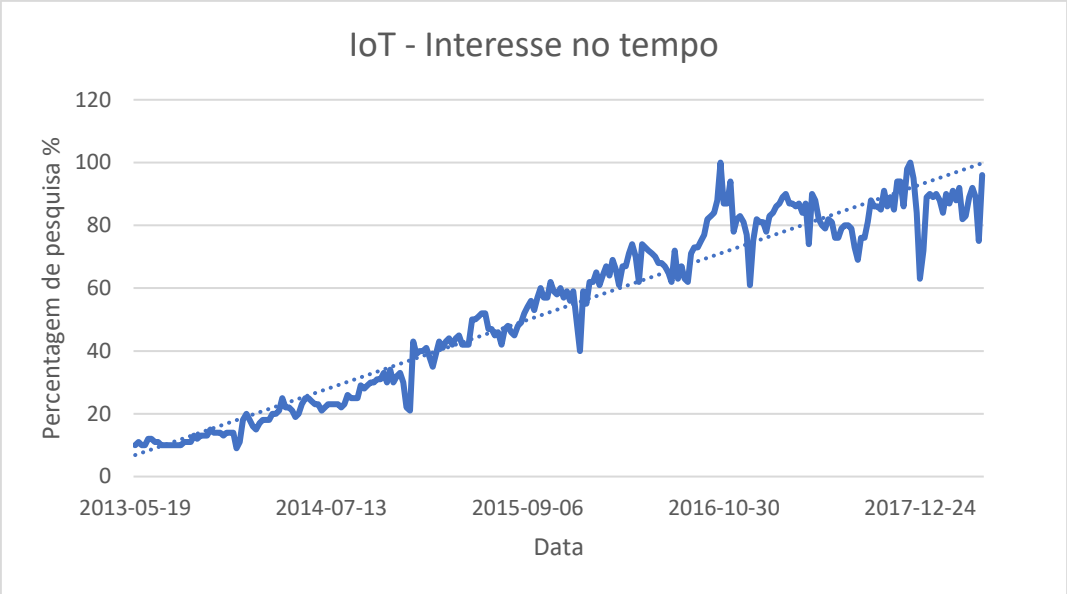

Figura 2.1 - Evolução no tempo do interesse de Internet das Coisas no Google Trends.

Também podemos visualizar estas estáticas representadas num mapa mundial de cidades, de acordo da localização das pesquisas:

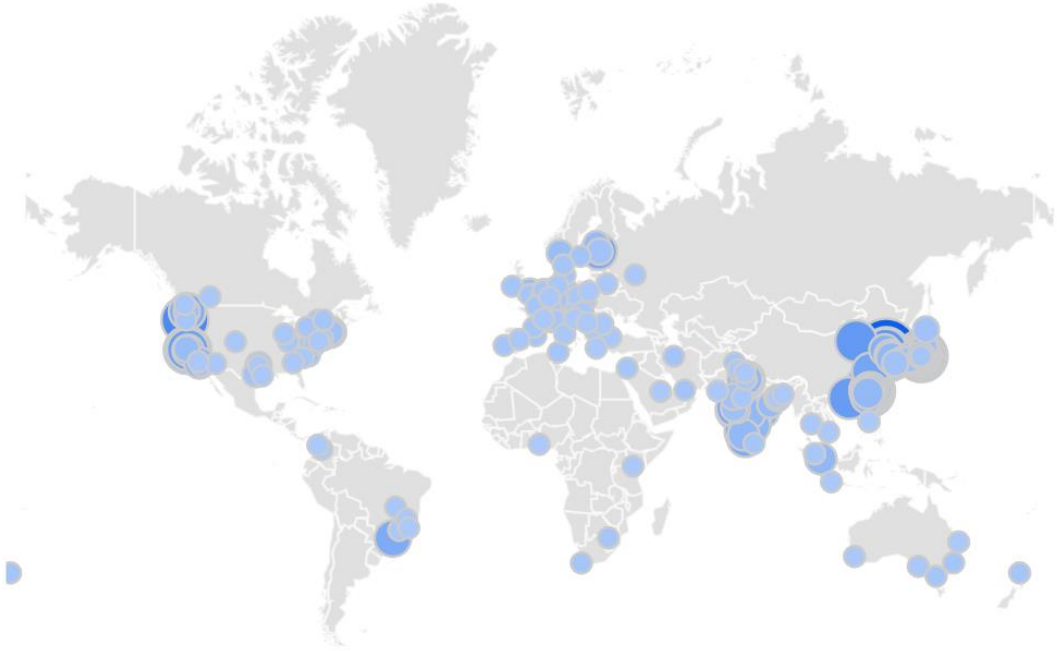

Figura 2.2 - Mapeamento geográfico de pesquisa do tema “IoT”.

Assim, podemos observar a importância deste assunto na Europa, que mesmo não sendo um dos pontos mais abundante em intensidade, é sem dúvida uma preocupação ativa na generalidade dos países europeus.

2.1.2 Impacto da Indústria 4.0

A indústria 4.0 tem como principal objetivo estimular e melhorar a produtividade das empresas aumentando a eficácia e a eficiência da produção, reduzindo falhas, custos de produção, aumentando a qualidade de produção, mas também criar novos modelos de negócio. As empresas procuram diferenciarem-se da concorrência, criando novos serviços de apoio ao cliente e inovar em novos produtos. No entanto já se verificam empresas que procuram personalizar os seus produtos de acordo às necessidades e preferências específicas de cada cliente. Por outro lado, os trabalhos repetitivos tendem a desaparecer, oferecendo oportunidade para criar novos negócios e novas funções com forte vertente criativa e tecnológica [17].

A Amazon é uma empresa de e-*commerce* que aplica algumas das tecnologias inerentes à indústria 4.0. Ao efetuar uma encomenda na Amazon, a aquisição é processada pelo servidor, verifica se dispõe dos elementos. Ao confirmar a compra, o(s) artigo(s) é recolhido por robôs que percorre o armazém trazendo as prateleiras com a encomenda ao funcionário. Ao longo de todo o processo, o artigo é verificado pelo robot, mas também pelo funcionário, de forma a evitar envio dum artigo errado. Ao enviar os artigos ao cliente, este tem a possibilidade de verificar o estado da compra.

2.1.3 Benefícios da Indústria 4.0

Devido a interligação das máquinas em rede, a monitoração da produção por meio de sensores, e graças à redução da intervenção humana nos processos de produção, as linhas de produção terão maior resiliência operacional, prevenindo falhas e paralisação da linha de produção. Manutenção não planeada causada por avarias resultando na paragem da linha de produção, pode ser evitada com a aplicação de diversas tecnologias [18]. Podendo ainda, graças a estas características, as próprias máquinas terem capacidade de reprogramação/reconfiguração remotamente, reduzindo a manutenção técnica no local [19]. Com a melhoria na eficiência da captação e geração de energia, é possível economizar no consumo energético e, mesmo assim, manter a alta produtividade, contribuindo para a sustentabilidade saudável do planeta.

Estes benefícios traduzem logicamente o objetivo natural da iniciativa da indústria 4.0, a redução de custos e otimização da produção.

Sumário de benefícios da indústria 4.0:

1. Melhor aproveitamento dos recursos
2. Redução de falhas
3. Consequente redução de custos de produção
4. Rapidez e flexibilidade de produção
5. Aproveitamento de logística
6. Eliminar ou reduzir funções perigosas e/ou difíceis ao humano

2.1.4 Política da Indústria 4.0

A Indústria 4.0 está presente na europa, pois é usada como motivação nas campanhas políticas. Mesmo em Portugal, verifica-se grande entusiasmo na aplicação da Indústria 4.0, e por isso incentiva-se a aposta das empresas a otimizarem seus serviços e sistemas de produção. Isto acontece porque com a difusão de tecnologia 4.0, acredita-se que a europa pode criar mais emprego e valor económico.

A Indústria 4.0 é um movimento que utiliza um conjunto de “novas” tecnologias, a qual se prevê uma adesão acentuada nos próximos anos. Tendo em conta que este movimento é considerado estar numa fase de difusão, Portugal tem boas condições de estruturar um novo modelo industrial, e criar novos produtos e serviços.

2.2 Linguagens de programação visual e ferramentas associadas

As linguagens de programação visual e ferramentas associadas (Visual Programing Languages - VPL) [8][20][21], permitem desenvolver programas manipulando figuras, símbolos, linhas, entre outros, evitando o uso de programação tradicional textual. O objetivo principal destas ferramentas é disponibilizar um ambiente de programação mais fácil, rápido e eficiente possível, de forma reduzir tempo e recursos associados ao desenvolvimento de projetos.

O uso de VPLs encontra-se num estado de crescimento pelo que já existem várias empresas e universidades focalizadas no desenvolvimento de diversos tipos de tecnologia para proporcionar novas soluções mais rapidamente e eficientemente para sistemas IoT. Este método de desenvolvimento é usado em diversas áreas tal como domótica, cidades inteligentes, automação, indústria, etc…

I - Tabela de Investigação e Desenvolvimento de VPLs.

VPL	Data de lançamento	Empresas envolvidas	Localização
Node-Red	2013	IBM	Nova Iorque, EUA
NETLab Toolkit	2014	Intel e ArtCenter	Califórnia, EUA
Ardublock	2011	Eclipse	Ontário, Canada
AT&T Flow Designer	2013	IBM	Nova Iorque, EUA
Reactive Blocks	2011	Norwegian University of Science and Technology, ECLIPSE e Bitreactive	Trondheim, Noruega
LabView	1986	National Instruments	Texas, EUA
Simulink	1984	MathWorks	Massachusetts, EUA.
IOTP-Tools	2012	GRES Research Group	Caparica, Portugal
IOPT-Flow	2016	GRES Research Group	Caparica, Portugal

2.2.1 Node-RED

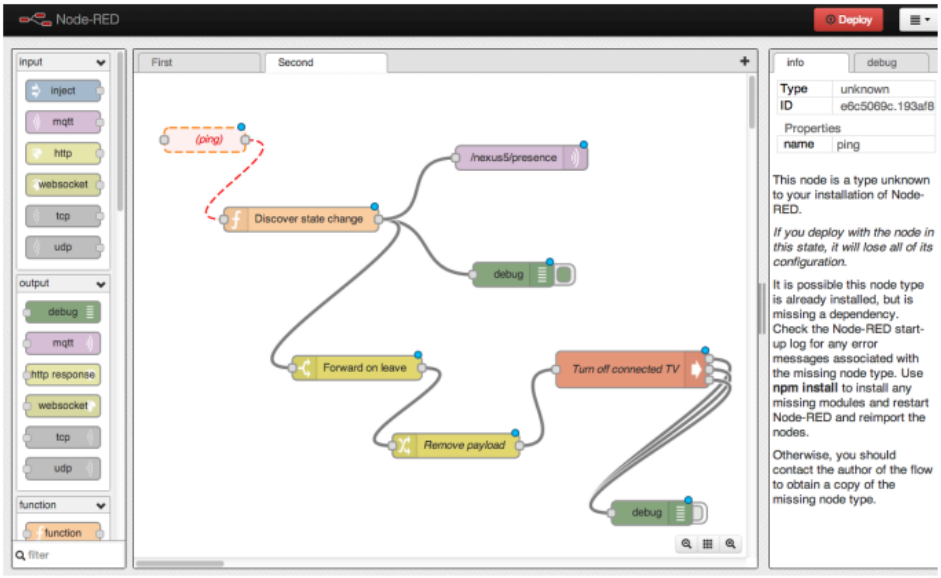

Figura 2.3 - Editor gráfico Node-RED.

Node-RED é uma ferramenta visual desenvolvida pela IBM [22], que permite conectar dispositivos e serviços online, a partir de um editor gráfico (Figura 2.3). A ferramenta permite ligar dispositivos IoT mas também programar eventos via HTTP, websokets, TCP, entre outros. Esta ferramenta baseia-se uma plataforma que possibilita controlar qualquer dispositivo IoT remotamente, permitindo alterar processos a partir da geração de um evento emitido remotamente a partir de um email, ou responder a eventos gerados por Twitter.

A conexão a partir do Node-RED consiste em desenhar ligações entre *nodes*, adicionando parâmetros a cada uma, a partir de código Javascript, de forma a particularizar o diagrama, permitindo um entendimento visual natural [23][24].

2.2.2 NETLab Toolkit

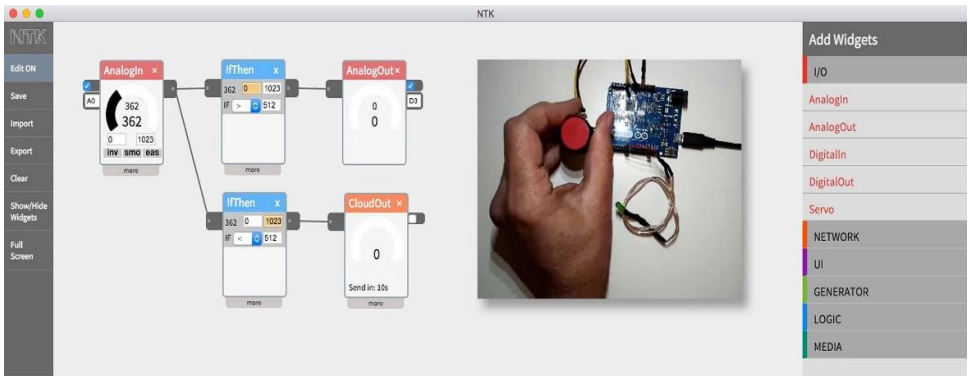

Figura 2.4 - Ambiente gráfico de NETLab Toolkit.

O NETLab Toolkit (Figura 2.4) [25] permite a modelação de sistemas a partir de blocos que podem ser arrastados e manipulados (“*Drag and Drop”*). Fornece uma interface web gráfica para conectar sensores e atuadores. NETLab Toolkit suporta diversos hardwares, tais como: Arduíno, Raspberry Pi, Intel Galileo e Arduíno Tre.

A modelação dos sistemas é realizada a partir de parametrizações de blocos e respetivas ligações entre blocos, mas também são usadas linguagens de programação (Node. js, HTML5 e JavaScript) para apoio na programação do projeto.

2.2.3 Ardublock

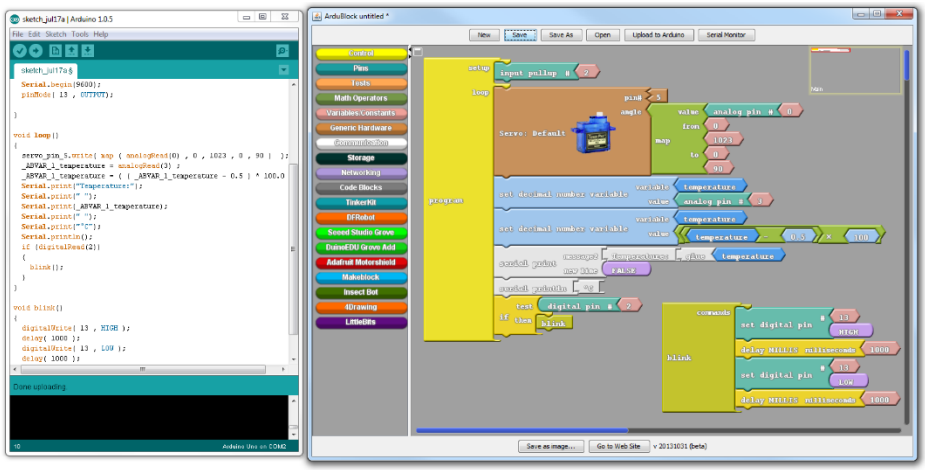

Figura 2.5 - Painel de desenvolvimento de Ardublock.

Ardublock [26] [27] (Figura 2.5) é uma plataforma *open source* que permite a criação de aplicações Arduíno a partir de um ambiente visual, ajudando o usuário a se conectar e visualmente

programar o Arduíno para aplicações IoT, sem a necessidade de conhecer linguagens de programação textual. É uma extensão para o Arduíno IDE e Eclipse IDE, permitindo a geração de código Arduíno e Java [28].

2.2.4 AT&T Flow Designer

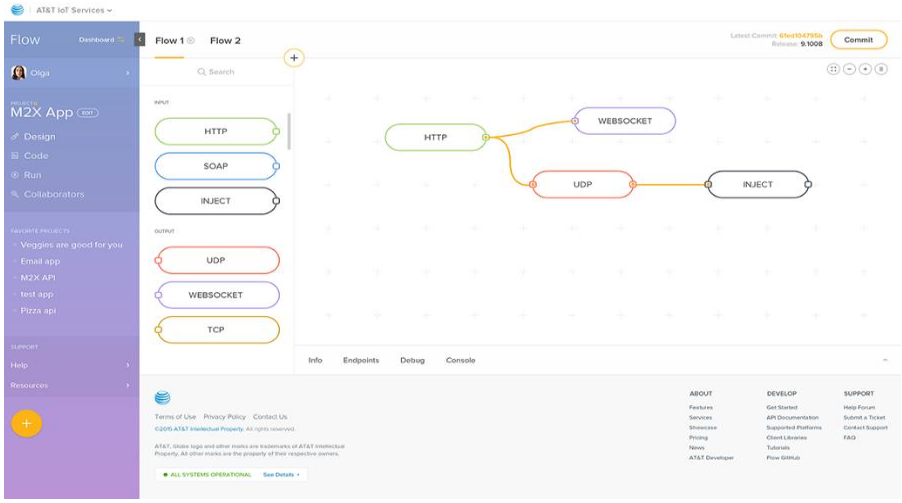

Figura 2.6 - Ambiente de AT&T Flow Designer.

AT&T Flow Designer (Figura 2.6) é uma extensão de Node-Red [29] [30], para armazenamento de dados analíticos baseado no método de edição do Node-Red. É uma ferramenta visual intuitiva que permite ao programador criar protótipos para dispositivos IoT com a capacidade de associar várias fontes de dados, métodos de comunicação, e dispositivos. Esta ferramenta permite reduzir o tempo de desenvolvimento de negócios e protótipos.

2.2.5 Reactive Blocks

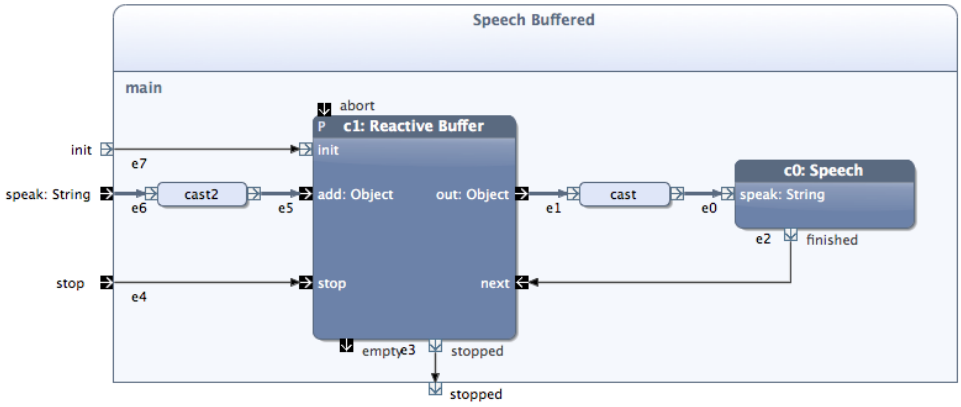

Figura 2.7 – Ambiente gráfico de um sistema Reactive Blocks.

Reactive Blocks (Figura 2.7) [31] [32] é uma ferramenta que permite a construção em blocos alternadamente com Java em Eclipse. Oferece um ambiente de desenvolvimento gráfico para suportar modelação de blocos e geração automatizada de código, permitindo desfrutar das capacidades de ligação IoT em tempo real [26], sem retirar a flexibilidade de programação textual ao programador e criar graficamente aplicativos de IoT reutilizáveis e complexos.

2.2.6 LabVIEW

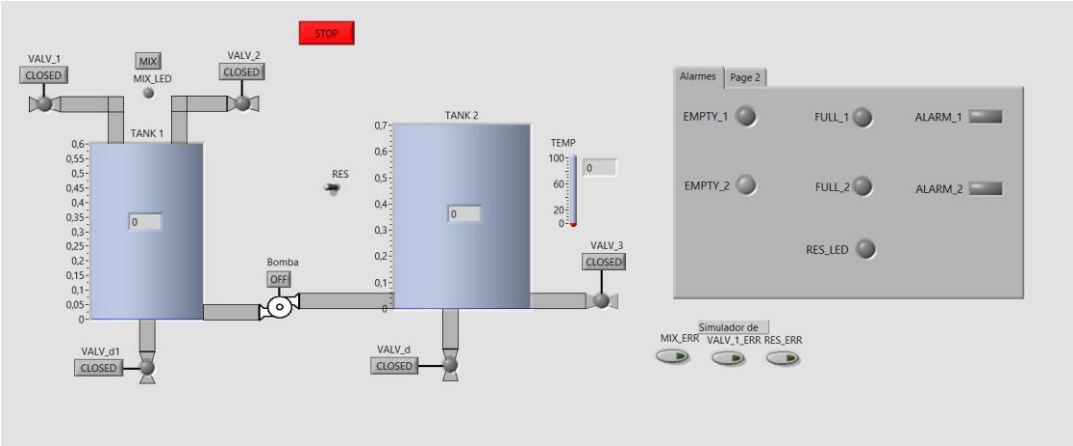

Figura 2.8 - Exemplo de um sistema de tanques no painel frontal do LabVIEW.

LabVIEW (Figura 2.8) é um laboratório virtual para instrumentos associados à engenharia industrial [38] [39]. A utilização do LabVIEW permite a criação de programas chamados VI, que se baseia no arrastamento de blocos que representam, e representam instrumentos de medição e

atuadores, tais como termómetros, monitores de medições, caldeiras, válvulas, diversos tipos de botões, motores, etc...

É mais fácil a implementação de sistemas de controlo, validando previamente o resultado dos sistemas graças à simulação visual do projeto, sem o custo e montagem do equipamento. Esta abordagem de programação, num sentido de investigação prévio do projeto, permite obter rapidamente e facilmente um protótipo virtual e resultados do comportamento do sistema.

Características principais do LabVIEW:

- Ambiente virtual
- Programação dataflow
- Capacidade de comunicação entre VIs
- Controlo Remoto
- Custo de licença pessoal

2.2.7 Matlab/Simulink

Matlab [33] é uma linguagem de alto nível associada a um IDE que permite analisar dados, desenvolver aplicações, criar modelos matemáticos e algoritmos numéricos com mais facilidade e eficiência do que com linguagens de programação tradicionais.

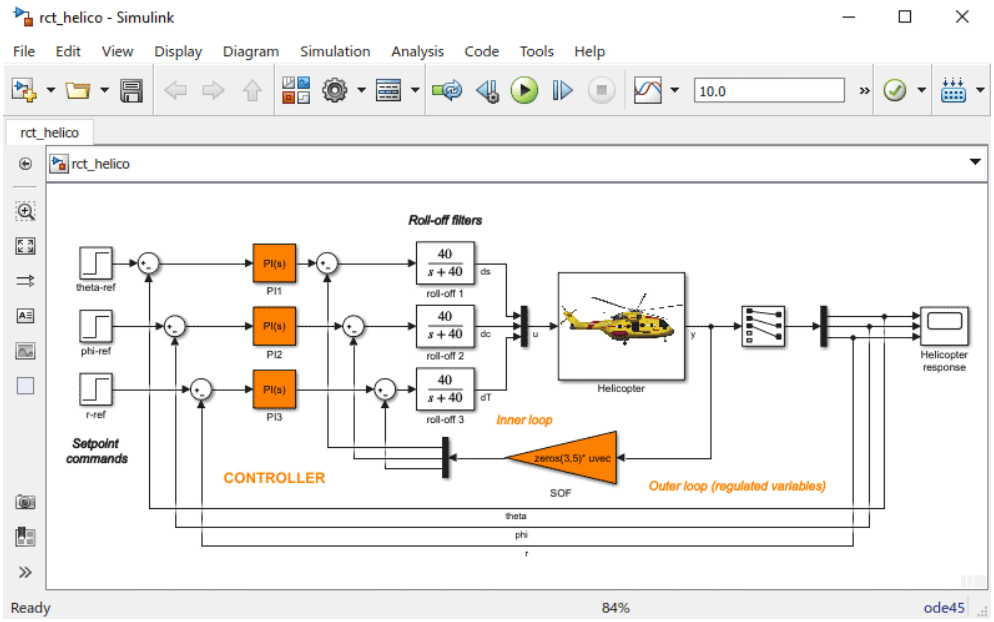

Figura 2.9 - Exemplo de modelação de sistema em Simulink.

Simulink (Figura 2.9) é uma ferramenta integrável com o Matlab [41] [42]. Esta ferramenta oferece um editor gráfico baseado em diagrama de blocos, suportando simulação de sistemas de controlo, geração automática de código (C/C++, HDL, PLC, GPU, .NET e Java) e verificação de sistemas embutidos.

Matlab e Simulink em conjunto, são ótimas ferramentas para modelação de sistemas de controlo complexos ou desenvolvimento de sistemas de automação, sobretudo usado em âmbito acadêmico, mas também em ambiente industrial. Suporta muitos formalismos de desenvolvimento, disponibilizando grande quantidade de modelos para rápido desenvolvimento.

Características principais:

- Ambiente de desenvolvimento
- Suporte de *dataflow* e Petri nets
- Ambiente de simulação gráfica
- Geração de código
- Aplicações matemáticas
- Comunicação cliente/servidor TCP/IP
- Custo de licença pessoal

Numa perspetiva de analise de dados e para teste, o Matlab e Simulink suportam sistemas IoT a partir do ThingSpeak [34]. Por exemplo, um Arduíno ou Raspberry Pi pode publicar informação para uma infraestrutura de nuvem para depois o Mathlab/Simulink processar os dados.

2.3 IOPT

As IOPT-nets (Input-Output Place-Transition nets) [35] são uma classe de rede Petri [36] que permite especificar sistemas de automação e sistemas embutidos, bem como sistemas distribuídos globalmente-assíncronos localmente-síncronos (GALS – Globally-Asynchronous Locally-Synchronous). As redes de Petri IOPT são suportadas por dois ambientes de desenvolvimento de automação de projeto, as IOPT-Tools e as IOPT-Flow, ambos disponíveis online e de acesso livre.

2.3.1 IOPT Net

As redes de Petri foram inventadas por Carl Adam Petri em 1939, com finalidade de representar processos químicos, apenas mais tarde foram documentadas na sua tese de doutoramento.

Redes de Petri são representações matemáticas para modelos de sistemas. O objetivo passa por representar sistemas a partir da modelação gráfica baseada em lugares, transições e arcos.

Os lugares podem acumular marcas, no entanto, apenas se pode disparar transições quando todos os lugares ligados à transição possuírem pelo menos uma marca. Transições são geradas por eventos do sistema, e quando se verifica uma transição, são consumidos as marcas de *input* para gerar novas marcas.

As IOPT-nets por sua vez baseiam-se em redes de Petri, admitem lugares, transições e arcos, no entanto adicionam a definição de sinais de input e output, eventos de input e output e arrays. Os sinais de input permitem obter informação externa de dispositivos, por leitura de portas de entrada. Por outro lado, os sinais de output permitem afetar o meio envolvente, manipulando as portas de saída do hardware.

As transições IOPT permitem ser parametrizadas de forma a condicionar os consumos e geração de marcas, estabelecer grau de prioridade de modo a evitar conflitos entre transições.

Caraterísticas das IOPT nets:

- Sinais de input/output
- Eventos de input/output
- Parametrização da rede Petri (condições de disparo de transições)

2.3.2 IOPT-Tools

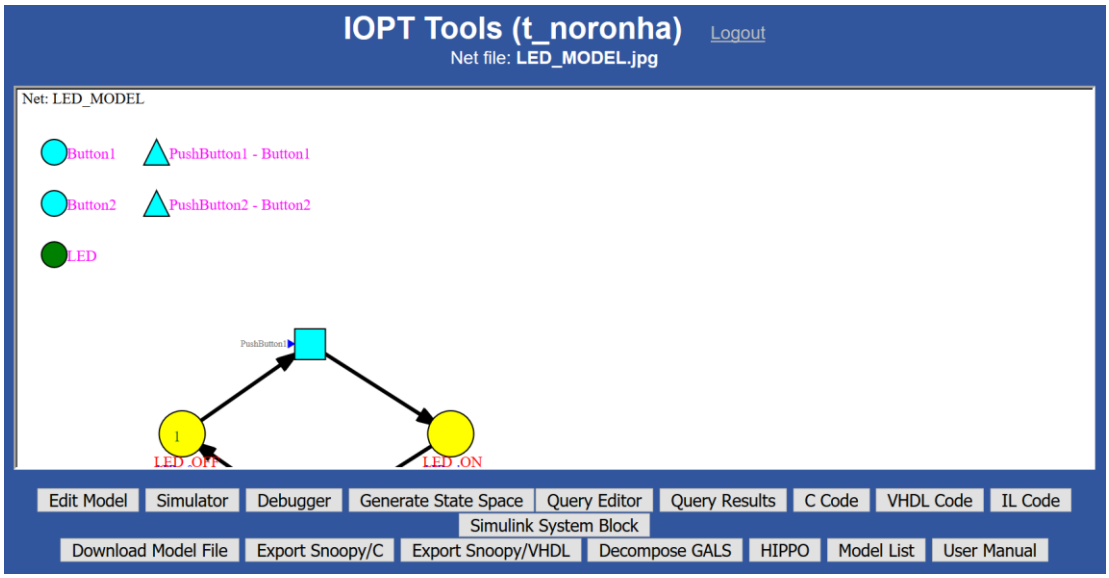

Figura 2.10 - Ambiente principal IOPT-Tools.

As IOPT-Tools (Figura 2.10) [37] permitem, entre outras coisas, especificar, validar e suportar a implementação de controladores embutidos. Possui um ambiente de desenvolvimento gráfico para a edição de modelos IOPT, bem como ferramentas para simular e verificar propriedades, através da geração e análise do espaço de estados do sistema. Dispondo de geradores automáticos de código, possibilita a geração de código "C" [38] e de código de descrição de hardware em VHDL [39]. Esta ferramenta Web permite ser utilizada em dispositivos com menor quantidade de recursos devido à sua baixa ocupação em memória (5 MBytes), reservando as tarefas de processamento menos intensivas para o *browser*, mas delegando tarefas mais exigentes ao servidor, como por exemplo a geração dos espaços de estado do modelo.

Características principais das IOPT-Tools:

- Ambiente de edição de um modelo
- Simulação de modelo
- Geração automática de código
- Monitorização do sistema (Debuguer)

2.3.2.1 Edição

As IOPT-Tools baseiam-se no formalismo de redes de Petri, e por isso do ponto de vista de modelação, o editor (Figura 2.11) contêm transições, marcas, arcos e lugares, mas de ponto de vista de representação de sinal encontramos blocos de sinais de output, sinais de input, ainda eventos de output e input. As IOPT-Tools disponibilizam de ferramentas de edição (Figura 2.12).

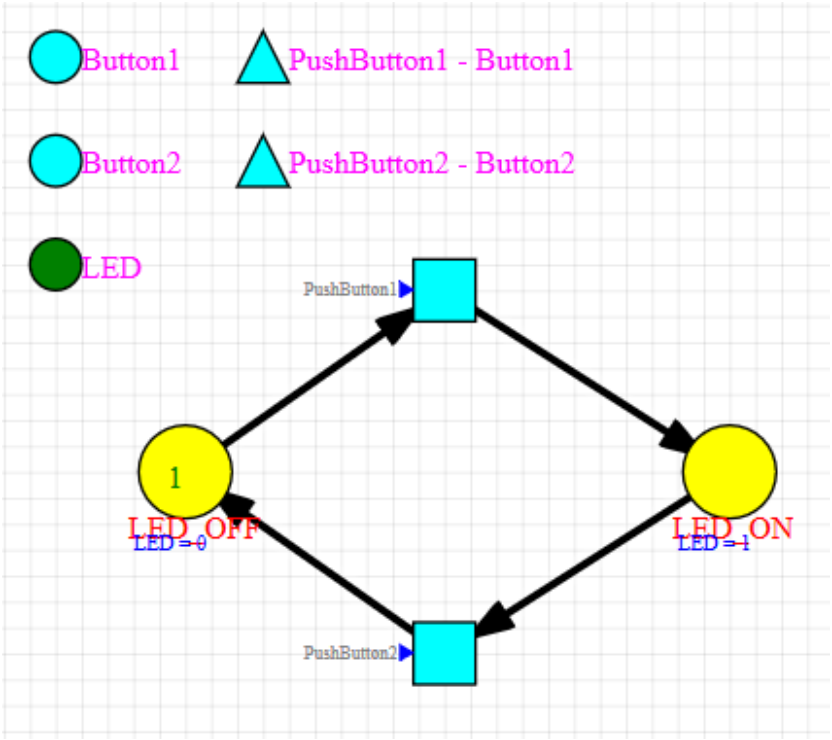

Figura 2.11 - Ambiente de edição das IOPT-Tools.

Figura 2.12 - Ferramentas de edição em IOPT-Tools.

Os sinais de input são usados para obter informação proveniente do meio físico do dispositivo (por exemplo pino do Arduíno ou Raspberry Pi, podendo definir no editor a porta do dispositivo), de forma a efetuar leituras do sistema (representado por "LED_ON" na Figura 2.11). No mesmo contexto temos os sinais de output, que permitem afetar atuadores, enviar sinais para sistemas externos ou para display de informação (representado por "Button1" e "Button2" na Figura 2.11).

Por outro lado, os eventos de input pretendem observar sinais de input e verificar variações, resultando num comportamento de *trigger*. De forma análoga, os eventos de output causam variações nos sinais de output.

Por sua vez, transições e lugares são diretamente ligados a estes sinais e eventos de forma a que o sistema tenha o comportamento desejado. No exemplo da Figura 2.11, podemos observar um sistema simples que controla um led a partir de dois botões externos. Quando o botão 1 é premido, o led acende, mas quando o botão 2 é premido o led desliga-se.

Para efetuar este comportamento, associou-se os lugares LED_ON e LED_OFF ao sinal de output LED de modo a exercer uma ação no led depois da transição transferir a marca para o lugar. Como indicado Figura 2.13, estabeleceu-se uma ação ao sinal de output "LED", sendo que quando o lugar LED_ON tiver uma marca, atribui-se ao sinal de output o valor "1".

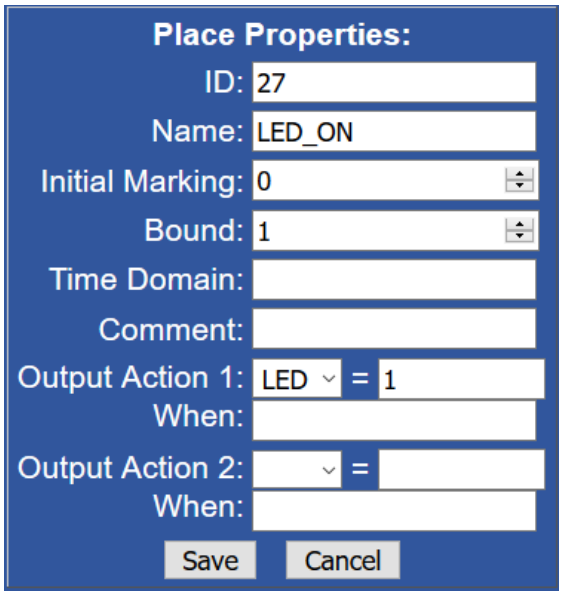

Figura 2.13 - Propriedades de um lugar IOPT.

Mas também se utilizaram sinais de input de tipo booleano, para a fazer a leitura externa de botões. Indicando o tipo, valor mínimo e máximo esperado, tal como o endereço da porta física.

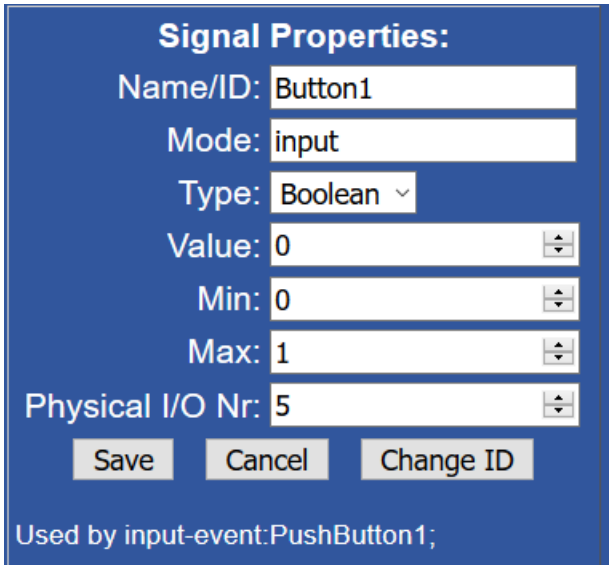

Figura 2.14 - Propriedades de um sinal IOPT.

Alem disso, pode-se aplicar um sinal evento para gerar apenas uma transição quando se verifica uma variação ‘0’ para ‘1’, e vice-versa.

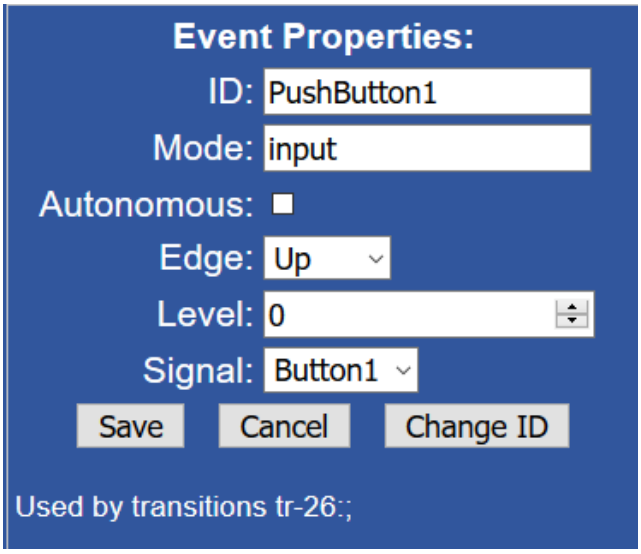

Figura 2.15 - Propriedades de um evento IOPT.

Assim, quando é gerado o evento “PushButton1”, e caso os requisitos de marcas serem preenchidas, este afeta a transição associada, resultando no disparo da transição, gerando novas marcas.

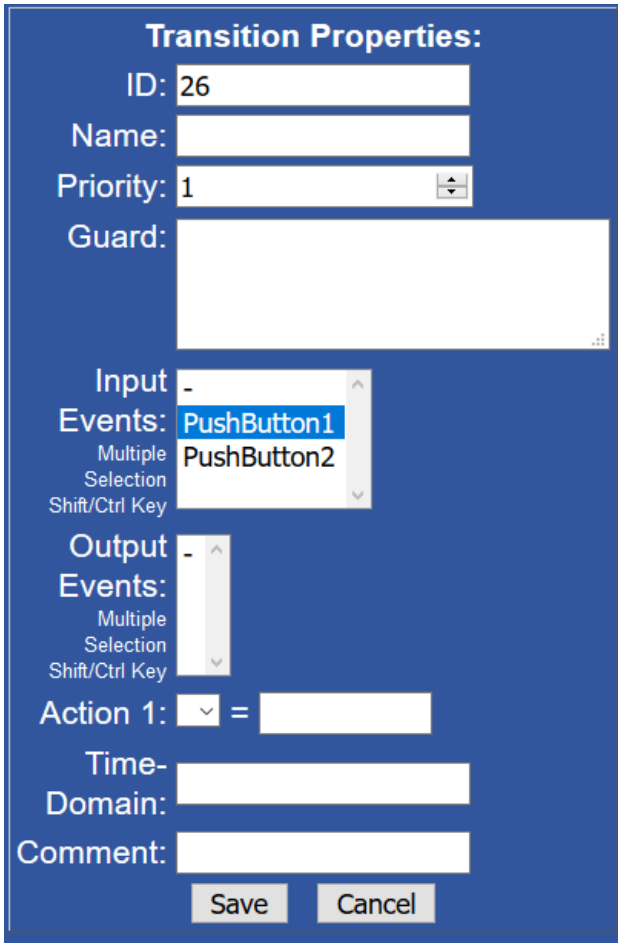

Figura 2.16 - Propriedades de uma transição IOPT.

2.3.2.2 Validação

2.3.2.2.1 Simulação

As IOPT-Tools dispõem de ferramentas de simulação que promovem a deteção de erros de edição na modelação dos modelos. A simulação é muito útil pois permite o teste do sistema/controlador sem que seja necessário a implementação do sistema na plataforma física, evitando o desperdiço tempo no upload do programa gerado para o hardware.

A simulação IOPT-Tools disponibiliza várias opções que permitam visualizar o comportamento do modelo. É possível forçar sinais de input para testar estados do sistema; *debug step by step*; guardar o histórico da simulação do sistema; e controlo de frequência de *clock*.

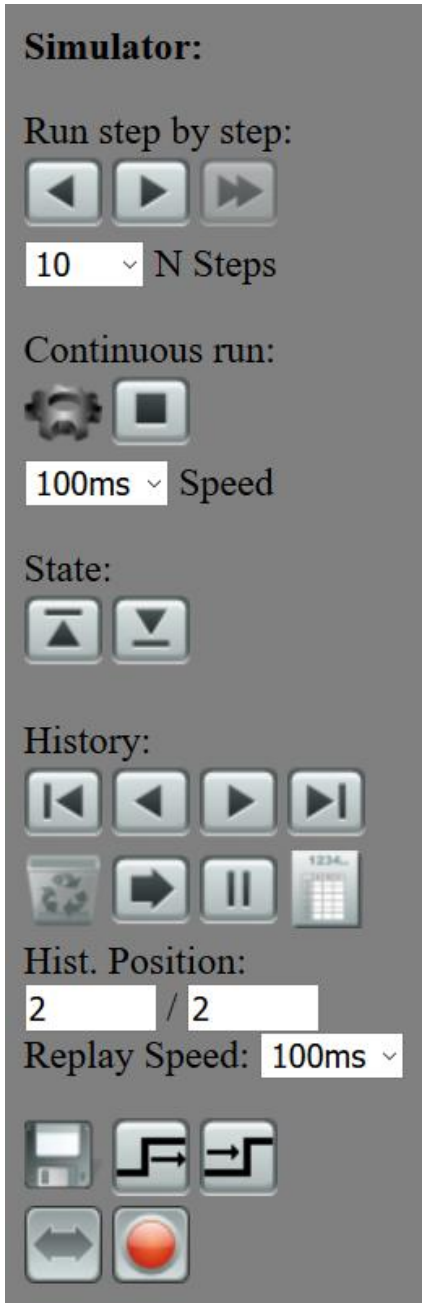

Figura 2.17 - Ferramentas de simulação IOPT-Tools.

2.3.2.2.2 Verificação (Gerador de espaços de estados)

Redes de Petri são geralmente associados aos espaços de estados e por isso, as IOPT disponibilizam uma ferramenta que permite fazer a verificação dos espaços de estados dos modelos de redes IOPT [40]. É possível detetar posições de *deadlock* ou *livelock* a com facilidade, poupando tempo de *debug* ao programador.

2.3.2.2.3 Monitorização (Debugger)

As IOPT-Tools disponibilizam uma ferramenta para monitorização de sistemas/controladores desenvolvidos na plataforma IOPT-Tools. O *debugger* permite monitorizar remotamente sistemas/controladores executados, e permite ainda forçar sinais de input e output remotamente de modo a provocar falhas no sistema. O acesso remoto é feito a partir de um login e uma password definida na compilação do programa.

2.3.2.2.4 Geração automática de código

As IOPT-Tools permitem a geração automática de código, em diversas linguagens de programação [9] [40], a partir de modelos IOPT:

- C, Arduíno
- VHDL
- JavaScript
- IL (Instruction List)

Figura 2.18 - Geradores de código nas IOPT-Tools.

O editor IOPT-Tools permite a geração automática de código C, VHDL, JavaScript para *Web browsers*, ou IL para aplicação de automação industrial em PLC (*Programmable logic controller*).

A geração automática de código evita a necessidade de escrever código manualmente, poupando tempo e recursos no desenvolvimento do projeto. A parametrização de pinos em Arduíno é completamente contruída nas IOPT-Tools. No entanto, alguns microcontroladores ou FPGA obriga o programador a alterar o código gerado manualmente.

O gerador de código C gera vários ficheiros:

1. **dummy_gpio** – Este ficheiro permite registar os valores de entrada e saída em ficheiro de texto.
2. **http_server** – Permite a transmissão de *sockets* para monitorização a partir do *Debuger*.
3. **linux_sys_gpio** - A utilização deste ficheiro gere a associação dos sinais de entrada e saída a pinos, permitindo correr modelos em sistemas Linux sem detenção de pinos entrada/saída (não Raspberry Pi).
4. **Makefile** – Ficheiro com instruções para compilação do projeto.
5. **net_dbginfo** – Permite fazer *debug* da rede, obter informação sobre os sinais, marcas e disparos de transições e também forçar entradas.

6. **net_exec_step -** Função que executa um passo de execução da rede Petri, afetando transições.
7. **net_functions -** São declaradas as funções de implementação e regras da rede de Petri IOPT.
8. **net_io.c** – Ficheiro que gere a leitura e escrita de sinais de entrada e saída.
9. **net_main** – Ficheiro *main* cíclico do projeto.
10. **net_server** – Ficheiro com funções associadas à comunicação HTTP da rede de Petri em Arduíno.
11. **net_types -** Definição dos tipos de dados para a marcação de cada lugar, sinais de entrada ou saída e eventos.
12. **raspi_mmap_gpio** – A utilização deste ficheiro permite usufruir dos pinos de entrada e saída do Raspberry pi.

2.3.3 IOPT-FLOW

As IOPT-Flow [41] [40] apresentam um conjunto de ferramentas semelhantes às IOPT-Tools, desde o editor de modelos até aos geradores de código, mas suportando um formalismo de modelação (DS-Pnets – *Data-flow*, *Signals and Petri nets*) que combina IOPT-nets com *data-flows*. Este formalismo possibilita a especificação de sistemas complexos através do uso de componentes, sendo que a parte controlo é habitualmente especificada com redes de Petri e a parte de dados com *data-flows*.

As IOPT-Flow permitem reutilizar modelos desenvolvidos e criar sistemas de sistemas, criação de modelos distribuídos capazes de comunicar entre eles por sinais de input-output do hardware ou via internet.

Características principais das IOPT-Flow:

- Extensão das IOPT-Tools
- Reutilização de modelos (Sistemas de sistemas)
- Comunicação entre modelos
- Monotorização e controlo (Debuguer remoto)
- Simulação e validação dos modelos

2.4 Comparação entre IOPT, Simulink e Labview

Matlab/Simulink e Labview são ferramentas bastante utilizadas pela comunidade académica de modo a criar protótipos e simular ambiente industrial. Suportam vários formalismos de modelação, e a comunidade associada disponibiliza múltiplas bibliotecas de modelos para desen-

volver novas aplicações. No entanto estas ferramentas são produtos com fins comerciais, requerendo licença para utilização e desenvolvimento de projetos não académicos. Labview foca-se em simulação de ambiente industrial, por isso matlab/simulink tem capacidade para suportar Labview no desenvolvimento de controladores e algoritmos matemáticos.

IOPT-Tools e Matlab/Simulink suportam formalismos adequados para modelação de controladores embutidos: diagramas de blocos, Petri nets, permitem ainda a geração automática de código, comunicação TCP/IP, entre outros. Estas tecnologias parecem competir por soluções muito semelhantes, no entanto visto que as IOPT-Tools permitem gerar automaticamente modelos para serem abertos em Simulink, estas ferramentas têm potencial para se complementam. Por outro lado, também é relevante comparar as IOPT-Tools com Labview devido às suas diferenças, Labview pretende desenvolver sistemas industriais numa perspetiva de simulação de protótipos, oferecendo visualização de componentes. No entanto não oferece outros tipos de formalismos, abrindo assim uma porta para complementação com outras ferramentas como IOPT-Tools ou Matlab/simulink.

II - Tabela comparativa de características das IOPT-Tools, Labview e Matlab/Simulink

Ferramenta	Meio de aplicação	Formalismos	Instalação	Espaço de instalação	Licença
IOPT-Tools	Indústria e controladores	Petri nets	Browser	5 Mbytes	Académico
Labview	Sector industrial	Blocos	Inteiramente no computador pessoal	5-8 Gbytes	Fins comerciais
Matlab/Simulink	Sector industrial e controladores	Fluxogramas, petri net, blocos	Inteiramente no computador pessoal	7-12 Gbytes	Fins comerciais

3 Arrowhead Framework

O *Arrowhead Framework* foi desenvolvido no âmbito do projeto europeu *Arrowhead Project*, continuando a ser desenvolvido agora no projeto Productive 4.0. O *Arrowhead Framework* utiliza princípios de arquitetura orientada a serviços (SOA - *Service-Oriented Architecture*).

O *Arrowhead Framework* permite a interação entre vários sistemas/dispositivos em rede. O software inclui serviços que suportam a interação entre aplicações de sistemas, como *Discovery*, *Authorization*, *Orchestration*, e *Service Register*. Como ilustrado na Figura 3.1, o *Arrowhead Framework* oferece serviços que permitem o registo (*Service Register*), a pesquisa (*Service Discovery*), orquestração (*Orchestration*) de sistemas/dispositivos e autorização (*Authorization*), que visa autorizar a partilha de informação entre consumidor e fornecedor. Para a utilização do *Arrowhead Framework*, um sistema que preste um serviço tem de se registar no *Service Registry*, e um sistema que consuma informação usa o *Service Discovery* e o *Orquestration* para obter o melhor fornecedor existente na rede.

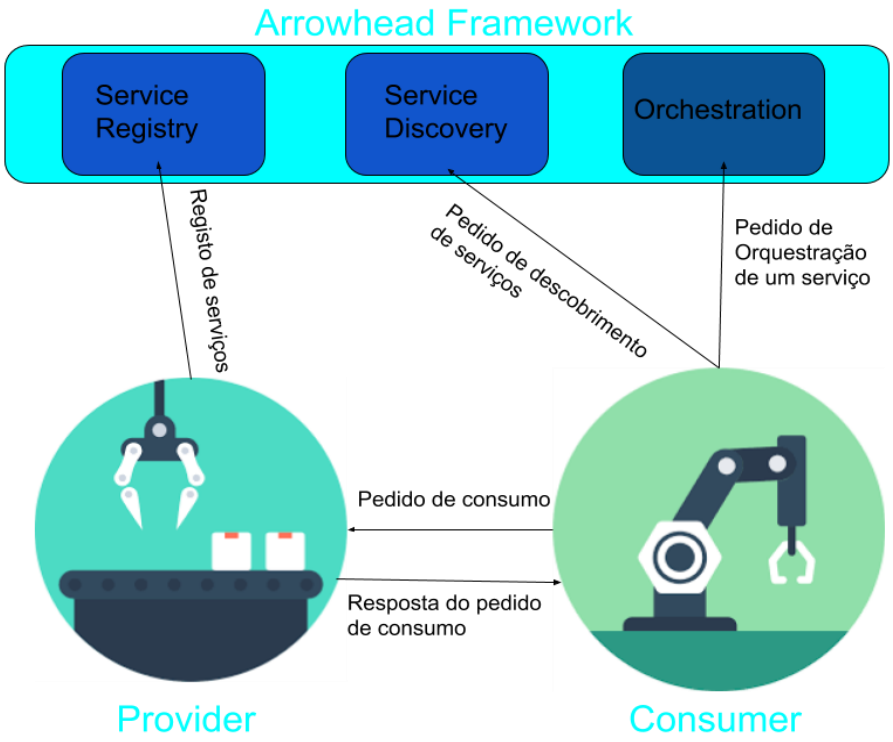

Figura 3.1 - Interação entre consumidor e fornecedor em Arrowhead Framework, adaptado de [42].

Quando um *consumer* verifica a necessidade de obter mais informação proveniente do ambiente exterior, pode recorrer ao *Arrowhead Framework* de forma a verificar se existe na rede, outro dispositivo (*provider*), que pode provisionar informação útil. Desta forma, o *provider* começa por se registar ao *Arrowhead Framework* fazendo um pedido de *service registry*, o *Arrowhead Framework* regista o serviço na base de dados e envia a confirmação de registo. A Figura 3.2 representa o processo de consumo e de fornecimento.

O *consumer* faz um pedido ao *Arrowhead Framework*, por sua vez o *Arrowhead Framework* verifica na base de dados a existência de serviços válidos para o *consumer*. Se valido, envia a confirmação de dados, juntamente com informação relativamente ao *provider ou aos vários providers* associados ao serviço. Agora que o *consumer* contém informação sobre o *provider*, imediatamente faz o pedido de consumo diretamente ao *provider* escolhido. Este processo é sequencial e eventual. No entanto, o *consumer* não necessita de fazer um novo pedido ao *Arrowhead Framework* para o mesmo serviço, visto que já contém dados sobre o *provider* do serviço, podendo então comunicar diretamente com este indefinidamente.

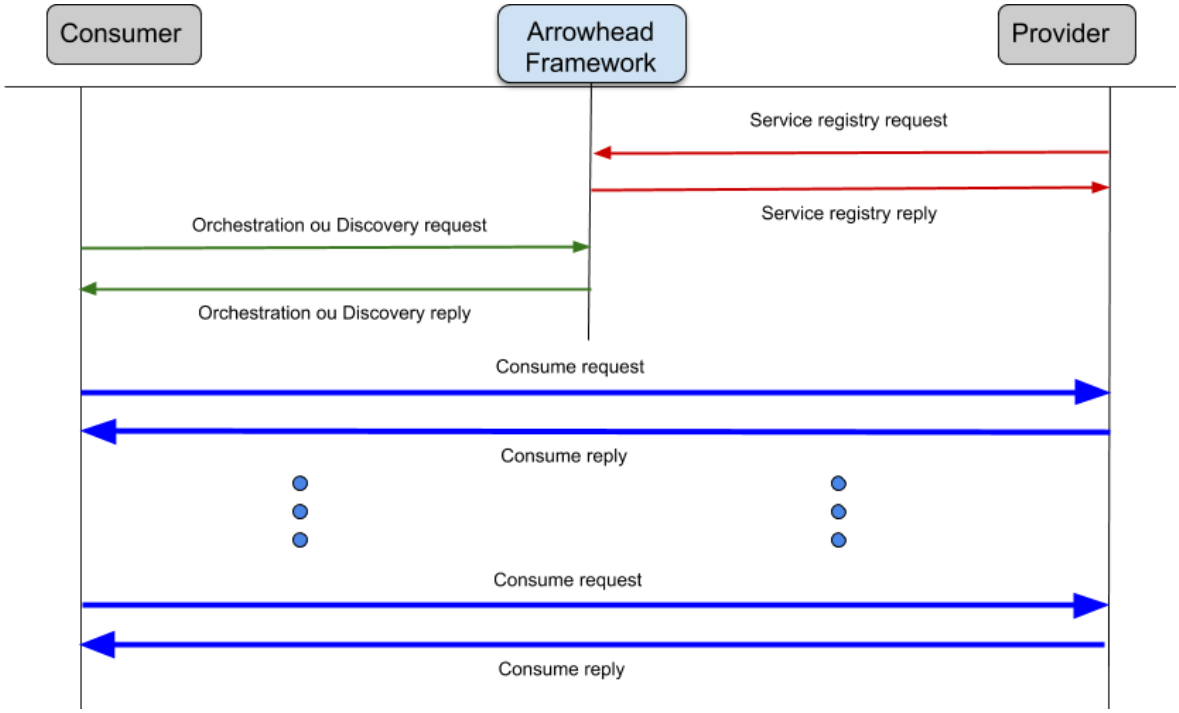

Figura 3.2 - Diagrama temporal do processo de consumo e fornecimento.

Tendo em conta a arquitetura do Arrowhead Framework, foram identificados três principais componentes: Servidor com o *Arrowhead Framework* core; Consumidor de serviços; e Fornecedor de serviços.

3.1 *Arrowhead Framework Core Systems*

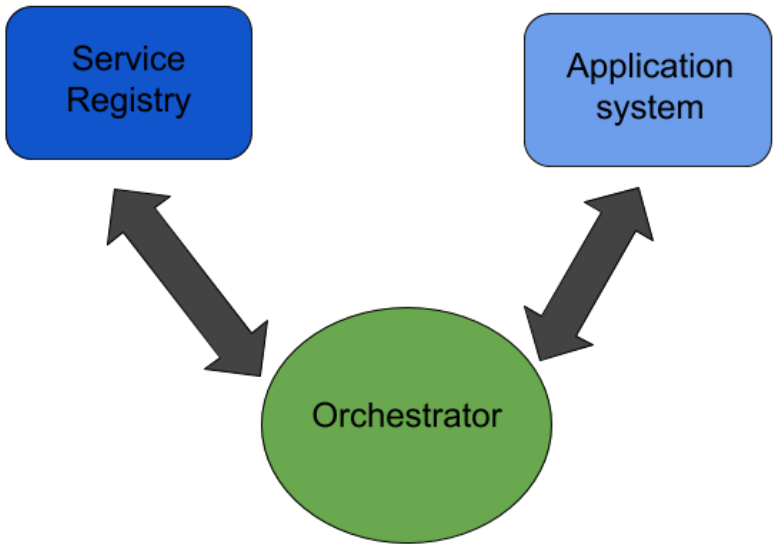

Figura 3.3 - *Arrowhead Framework* core systems.

Os principais sistemas (*Arrowhead Framework Core Systems*) disponibilizados pelo *Arrowhead Framework* são:

- *Service Registry* – É responsável pela receção de registos de serviços, tal como o respetivo registo na base de dados e disponibilização das listas de serviços disponíveis na base de dados.
- *Orquestrator* – Este serviço é responsável por analisar todos os fornecedores de serviços disponíveis e escolher o serviço mais adequado.
- *GateKeeper* – Este sistema pretende permitir *inter-cloud orquestration*
- *Gateway* – Este sistema é responsável em conectar *aplication systems* de *Arrohwead Framewoks* diferentes.
- *Authorization* – Providencia autorização ao Orquestrator e Gatekeeper para suas disponibilizarem seus serviços.
- Aplication System – Sistema que pode ser *provider* e/ou *consumer* de serviços

Os *Cores Systems* do *Arrowhead Framework* são acessíveis individualmente, e por isso, é permitido desativar os sistemas que possam não interessar. A seguir, vai ser descrito o processo de registo do *provider*, orquestração do *consumer* e o consumo de serviço detalhamente.

Como indicado anteriormente, para que o *consumer* utilize dados de sensores externos ao seu sistema, precisa de saber qual o melhor *provider* para fornecer a informação desejada. Para isso, o *consumer* envia um POST *request para o* orquestrator com um payload. O *orquestrator*

comunica com o *Service registry core* para verificar todos os serviços disponiveis, escolhe o mais indicado e responde ao *consumer*. Finalmente, o consumer faz um pedido ao *provider*. A Figura 3.4 repesenta o processo de consumo e fornecimento com a interação entre os *cores* do Arrowhead Framework.

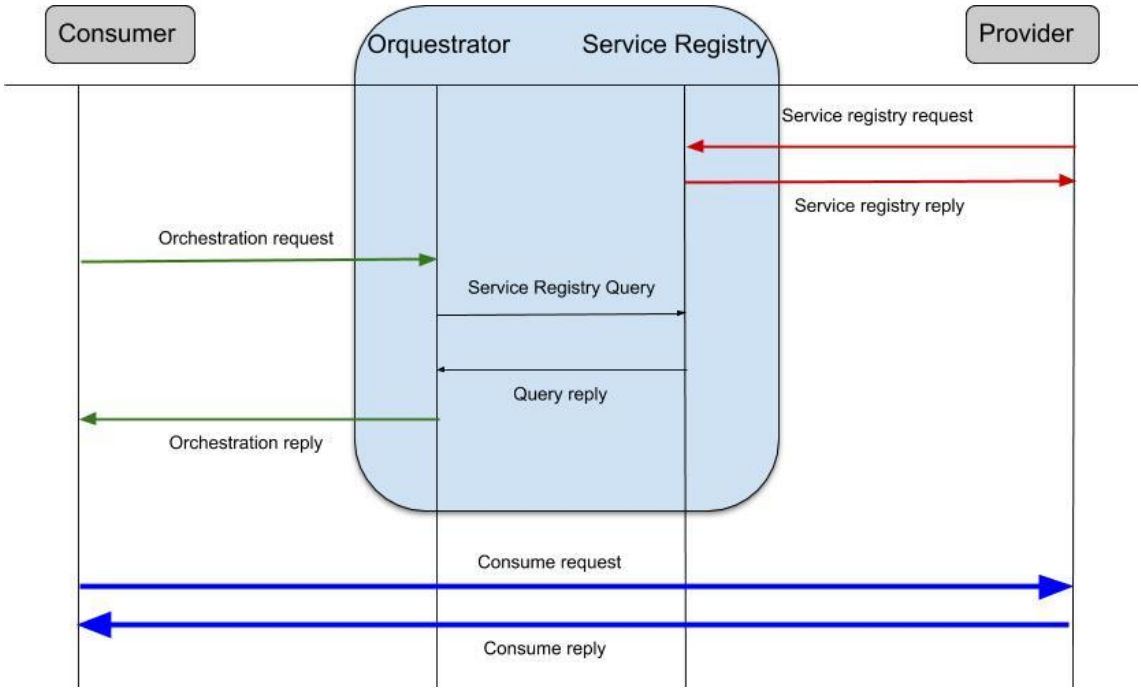

Figura 3.4 - Diagrama temporal do processo de consumo e fornecimento.

3.1.1 Service Registry

Tal como sugerido, o *service Registry* permite que os *providers* possam se registar na base de dados do *Arrowhead Framework*, associando serviços de fornecimento de dados internos do sistema.

O *Service Registry é responsável por:*

- *Registar e remover serviços*
- *Limpar os serviços offline da base de dados*

Quando o dispositivo *provider* se liga a rede, este regista-se ao *Arrowhead Framework* juntamente com os serviços disponíveis, enviando uma mensagem para o endereço do Service Registry do *Arrowhead Framework* (por exemplo http://127.0.0.1:8442/serviceregistry/register). O *Service Registry* encarrega-se de guardar na base de dados a informação sobre os sistemas, quais os serviços associados a cada sistema, e lista de interfaces e metadata, recebendo essa informação numa mensagem *payload* em formato JSON.

Relativamente ao serviço:

- *serviceDefinition* – Nome descritivo do serviço
- *unit* – Unidades das medições
- *serviceURI* – Localização do serviço no *provider* (diretoria)

Relativamente ao *provider*:

- *systemName* – Nome do sistema
- *address* – Endereço IP
- *port* – Porto

Exemplo de um *payload* de registo de serviço:

```
{
  "providedService": {
    "serviceDefinition": "IndoorTemperature",
    "interfaces": [
      "json"
    ],
    "serviceMetadata": {
      "unit": "celsius"
    }
  },
  "provider": {
    "systemName": "InsecureTemperatureSensor",
    "address": "127.0.0.1",
    "port": 8454
  },
  "port": 8454,
  "serviceURI": "temperature",
  "version": 1,
  "udp": false,
  "ttl": 0
}
```

Os *providers* podem registar vários serviços, ou várias instâncias do mesmo serviço, por isso quando o *consumer* faz o pedido de consumo, este envia qual o serviço que pretende, o *provider* analisa o pedido, identifica o serviço e responde. O *provider* deve identificar qual é o serviço que o *consumer* pretende, por isso o *ServiceURI* (o tipo de serviço) deve ser registado na aplicação *provider*, responsabilizando o *provider* por estabelecer o direcionamento para o *ServiceURI*.

Por outro lado, o *Service Resgistry* disponibiliza também o serviço *ServiceDiscovery*, que permite devolver a listagem dos serviços registados na base de dados. Isto permite ao *Consumer* fazer um pedido de *ServiceDiscovery* e ser-lhe atribuído uma lista com todos os serviços disponíveis com o *serviceDefinition*, no entanto, deixando o *consumer* analisar e escolher o serviço. Este serviço disponibilizado pelo *Service Registry* pode ser interessante se não existir um critério de avaliação válido para que o *Orquestrador* realize a escolha do melhor serviço, ou também quando o *consumer* necessita, por exemplo, de várias medições de vários *providers*.

3.1.2 Orquestrator

O *orquestrator* é o *sistema* responsável por comunicar e gerir os restantes sistemas do *Arrowhead Framework*. A sua principal função como moderador, é atribuir serviços adequados ao pedido de *orquestration* (serviço de *orquestration).* Este processo é iniciado pelo *Application system (consumer)* quando este faz um pedido de serviço em formato JSON, descrevendo e indicando qual o pedido de serviço (*Orquestration* ou *Discovery*). O *orquestrator* comunica com o *service registry* para extrair a listagem de serviços disponíveis para aquela descrição, escolhe o melhor e responde ao *consumer* com detalhes sobre o serviço, de modo a que se possa conectar.

O *orquestrator* deve receber do *consumer* um conjunto de parâmetros, dos quais se destacam:

- *systemName* – Nome/identificação do *consumer*
- address – Endereço ip
- *port* – Porto

Deve receber ainda, relativamente ao serviço que o *consumer* pretende consumir:

- serviceDefinition – Definição/tipo de serviço
- unit – Unidades
- interfaces – Formato de dados da mensagem
- metadata – Informações sobre dos dados

Exemplo de um payload enviado pelo *consumer* ao *orquestrator*:


```
{
  "requesterSystem": {
    "systemName": "client1",
    "address": "localhost",
    "port": 0,
    "authenticationInfo": "null"
  },
  "requestedService": {
    "serviceDefinition": "IndoorTemperature",
    "interfaces": [
      "json"
    ],
    "serviceMetadata": {
      "unit": "celsius"
    }
  },
  "orchestrationFlags": {
    "onlyPreferred": false,
    "overrideStore": true,
    "externalServiceRequest": false,
    "enableInterCloud": true,
    "enableQoS": false,
    "matchmaking": false,
    "metadataSearch": true,
    "triggerInterCloud": false,
    "pingProviders": false
```

```
        },
        "preferredProviders": [],
        "requestedQoS": {},
        "commands": {}
}
```


Por fim o *orquestrator* responde ao *consumer* enviando informação sobre *provider* e o serviço que o *provider* fornece:

- systemName – Nome do *provider*
- address – Endereço do *provider*
- port – Porto do *provider*
- serviceURI – Diretoria do serviço
- serviceDefinition – Definição do serviço
- unit – Unidades

Exemplo do *payload* enviado pelo *Orquestrator*:


```
        {
          "response":       [{
                    "service":          {
                              "serviceDefinition":"IndoorTemperature",
                              "interfaces":["json"],
                              "serviceMetadata":{
                                        "unit":"celsius"}
                                        },
                    "provider":         {
                              "systemName":"InsecureTemperatureSensor",
                              "address":"127.0.0.1",
                              "port":8454,
                              "authenticationInfo":null
                    },
                    "serviceURI":"temperature",
                    "instruction":null,
                    "authorizationToken":null,
                    "signature":null
                    }]
}
```


Para que o *orquestrator* possa escolher o melhor serviço disponível para o *Aplication system*, é possível a introdução de regras. Esta característica faz com que possam existir múltiplos *orquestrators* diferentes com os mesmos serviços registados, mas devolvendo resultados diferentes, pois os *consumers* devem receber informação de *providers* diferentes dependendo da conveniência e lógica de cada sistema.

As regras de *orquestration* podem ser introduzidas manualmente no *Orquestration system* pelo serviço *OrquequestrationManagement service*, ou então a partir de ferramentas externas como o *AutomationML*, que permite criar uma um plano descritivo do sistema.

3.1.3 Application System

Aplication System corresponde aos programas que representam os *consumers* e *providers*. Estas aplicações são desenvolvidas separadamente dos restantes *Arrowhead Framework Core systems*, dependendo da aplicação do projeto.

3.1.3.1 Consumer

A aplicação do sistema de consumo deve ser responsável por:

1. Detetar necessidade de consumo
2. Enviar um pedido de *orquestration* ou *serviceDiscovery*
3. Receber resposta do *orquestrator* ou *service registry*
4. Enviar o pedido de consumo ao *provider selecionado*
5. Receber resposta com a medição

Por sua vez, quando o *consumer* já conhece o *provider*, seja dado pelo *service registry* ou pelo *orquestrator*, basta fazer um pedido de consumo.

Exemplo de um *payload* de um pedido de consumo:

```
{
  "response":     [{
        "service":      {
              "serviceDefinition":"IndoorTemperature",
              "interfaces":["json"],
              "serviceMetadata":{
                    "unit":"celsius"}
                    },
        "provider":     {
              "systemName":"InsecureTemperatureSensor",
              "address":"127.0.0.1",
              "port":8454,
              "authenticationInfo":null
        },
        "serviceURI":"temperature",
        "instruction":null,
        "authorizationToken":null,
        "signature":null
        }]
}
```

Os *consumers* podem ter um comportamento eventual, ou seja, ser disparado por outro programa e seguir uma sequência finita de estados (Figura 3.5), ou então podem não ter a necessidade de voltar a fazer pedidos de *orquestration* ou *serviceDiscovery*, voltando a comunicar diretamente com o *provider (*Figura 3.6), pois o pedido de consumo pode ser repetido indefinidamente.

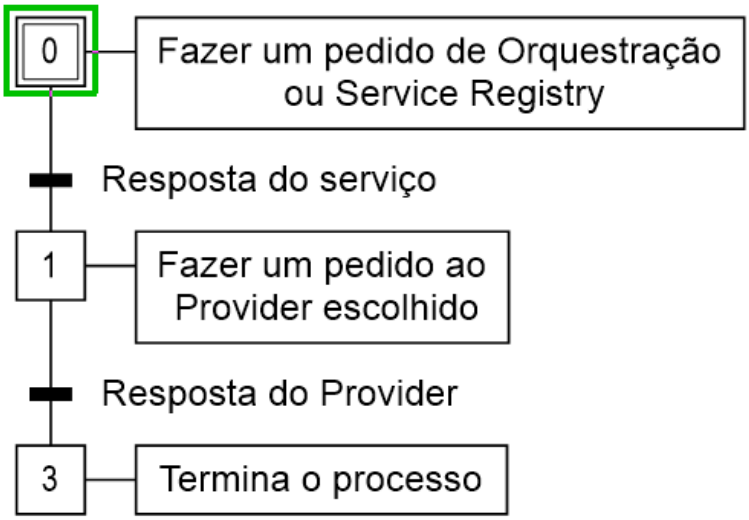

Figura 3.5 - Gracet do processo de um *consumer* sem repetição de consumo.

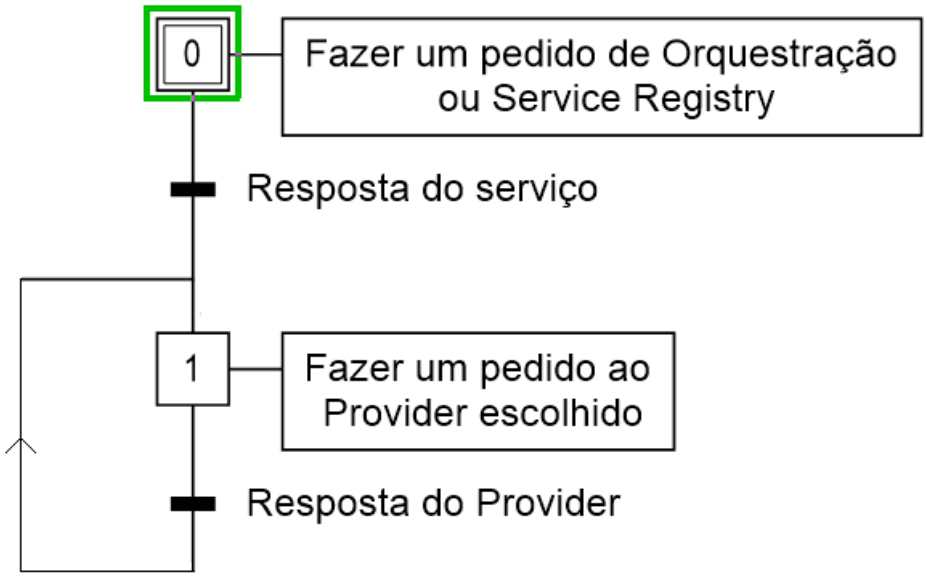

Figura 3.6 - Gracet do processo de um *consumer com repetição de consumo*.

3.1.3.2 Provider

O *provider* deve ter capacidade de:

1. Registar os seus serviços ao *service registry*
2. Esperar pedidos de consumo

3. Responder ao pedido de consumo

O *Aplication system provider* começa por se registar no *Arrowhead Framework* de modo a indicar os serviços que pode oferecer. No entanto no que toca ao consumo, o *provider* comporta-se como um *server*, apenas respondendo individualmente aos pedidos de consumo (Figura 3.7), no entanto é apenas uma possibilidade.

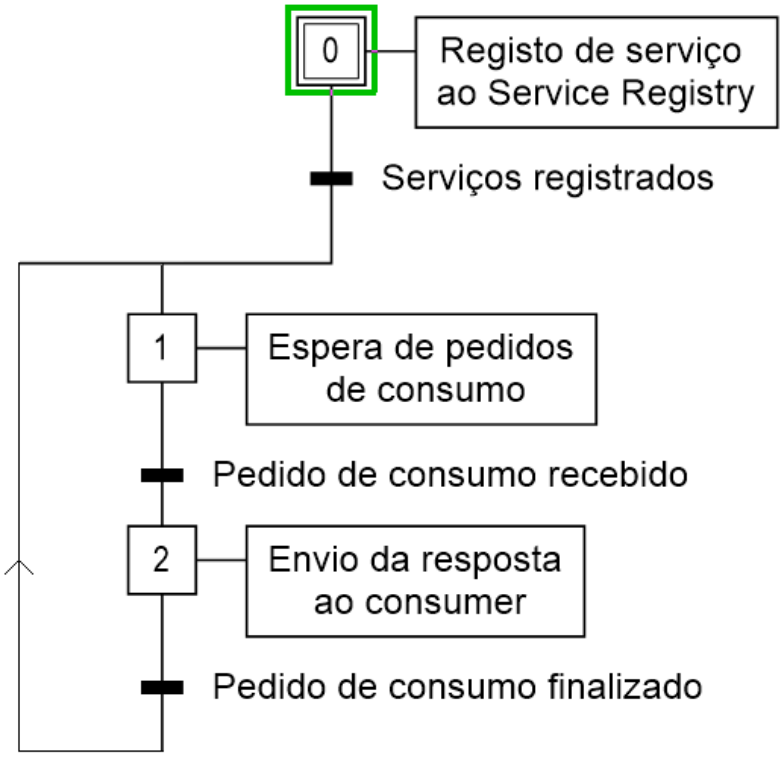

Figura 3.7 - Grafcet do processo de um *provider*.

Exemplo de um *payload* de resposta ao pedido de consumo:

```
{
   "bn": "TemperatureSensors_InsecureTemperatureSensor",
   "bt": 1506943344134,
   "bu": "celsius",
   "e": [
            {
                     "n": "Temperature_IndoorTemperature",
                     "t": 1506943344134,
                     "v": 21
            }
   ],
   "ver": 1
}
```

3.2 Comunicação HTTP (POST/GET)

O *Arrowhead Framework core system* tem uma abordagem de comunicação sobre HTTP (*Hypertext Transport Protocol)* por envio de mensagens em formato JSON.

Para poder consumir um serviço, o *consumer* comunica a partir de um HTTP POST *orquestration request* para o orquestrador, enviando um *payload* JSON. É esperado receber uma resposta também em formato JSON, com informação relativamente ao melhor fornecedor disponível para o serviço requisitado, assim, o *consumer* envia uma mensagem para o *provider* escolhido, esperando resposta com a medição desejada.

Do lado do fornecedor, o mesmo método de comunicação é valido, tanto para registo de serviço como para responder ao *request* (este deve esperar por pedidos de consumo e responder).

Qualquer componente que se registe ao *Arrowhead Framework* deve completar especificações, isto é, deve fornecer informação relevante para o sistema, é necessário que tando o *provider* como o *consumer* tenham o nome do serviço a registar/consumir e o endereço do *Arrowhead Framework* ao qual se pretende ligar. No entanto, na comunicação entre *consumer* e *provider*, não é obrigatório que usem o mesmo protocolo de comunicação. O *Arrowhead Framework* disponibiliza tradutores de forma a que *consumers* e *providers* comuniquem autonomamente a partir de "linguagens" diferentes. Assim, poderá vir a ser permitido a utilização de vários protocolos de comunicação (HTTP, MQTT, XMPP, CoAP, OPC UA …) [43].

4 Projeto biblioteca C

Durante o estudo realizado ao *Arrowhead Framework* foi testado um projeto java disponível, que implementa exemplos de *Aplication Systems*, que suportam a interação entre si e com o *Arrowhead Framework*, permitindo por exemplo registar e procurar serviços. No entanto, tendo em conta que as IOPT-Tools geram código C (não geram código Java), foi conveniente a criação de uma biblioteca C que suportasse a interação de *Application Systems* com o *Arrowhead Framework*. Tendo em conta que se pretendia que o código C, automaticamente gerado pelas IOPT-Tools, usasse essa biblioteca, foi necessário preparar uma biblioteca C que fornesse funções de *provider* e *consumer* preparadas para serem chamadas diretamente pelo código gerado pelas IOPT-Tools.

A biblioteca desenvolvida é dividida em duas partes: *provider* e *consumer*. Esta biblioteca C *Arrowhead Framework* contém funções de ambas as aplicações, mas completamente independentes.

A parte do *consumer* disponibiliza funções que permite ao utilizador rapidamente adaptalas à aplicação. Existem duas rotinas principais: *setup* e *loop*. O *setup* serve para realizar o pedido de orquestração para que depois este processo não seja repetido no *loop*. No ciclo, o programa efetua (sempre que conveniente) o pedido de consumo diretamente com o *provider*.

Inicia-se o programa no *setup* onde se prepara o consumo do serviço ao fazer o pedido de orquestração ao orquestrador. Uma mensagem em formato JSON é preparada e enviada para o *Arrowhead Framework* definido pelo utilizador, e com isto o programa guarda os dados do *provider* e prossegue para o consumo (Figura 4.1).

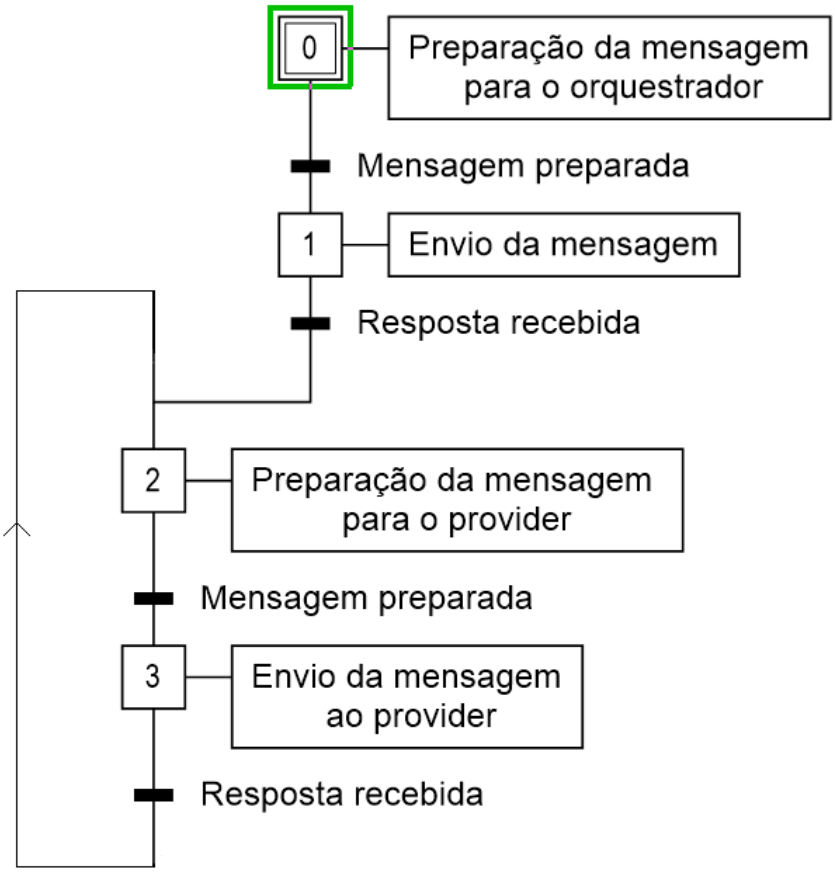

Figura 4.1 - Grafset da parte do consumidor.

Este processo é sequencial, podendo ser repetido entre o estado 2 e 3, no entanto é permitido a reorquestração de tempo em tempo para que a aplicação atualize o *provider* selecionado, ou para o caso do *provider* deixar de responder.

Pseudocódigo exemplo do *consumer*:

```
public provider = void;
main(){
        setup();
        cilco(NoT end){
            loop();
        }end
}

Setup(){
        mensagem = PreparaçãomensagemJsonParaOrquestrador();
        respostaMsgOrquestrador = EnviarMsgParaOrquestrador(mensagem);
```

```
        provider = HandleResposta(respostaMsgOrquestrador);
    }

    Loop(){
        mensagem = PreparaçãomensagemJsonParaProvider();
        respostaMsgProvider = EnviarMsgParaProvider(mensagem);
        Valor = HandleResposta(respostaMsgProvider);
        FazerCoisasComValor(Valor);
    }
```

Do mesmo modo, a parte do *provider* é dividida nas duas partes, *setup* e ciclo. Inicialmente no *setup*, o *provider* regista os serviços no *Service Registry* para não se repetir no ciclo. Completado o *setup*, o *provider* espera um pedido de consumo, até responder a um pedido e voltar para o estado de espera (Figura 4.2).

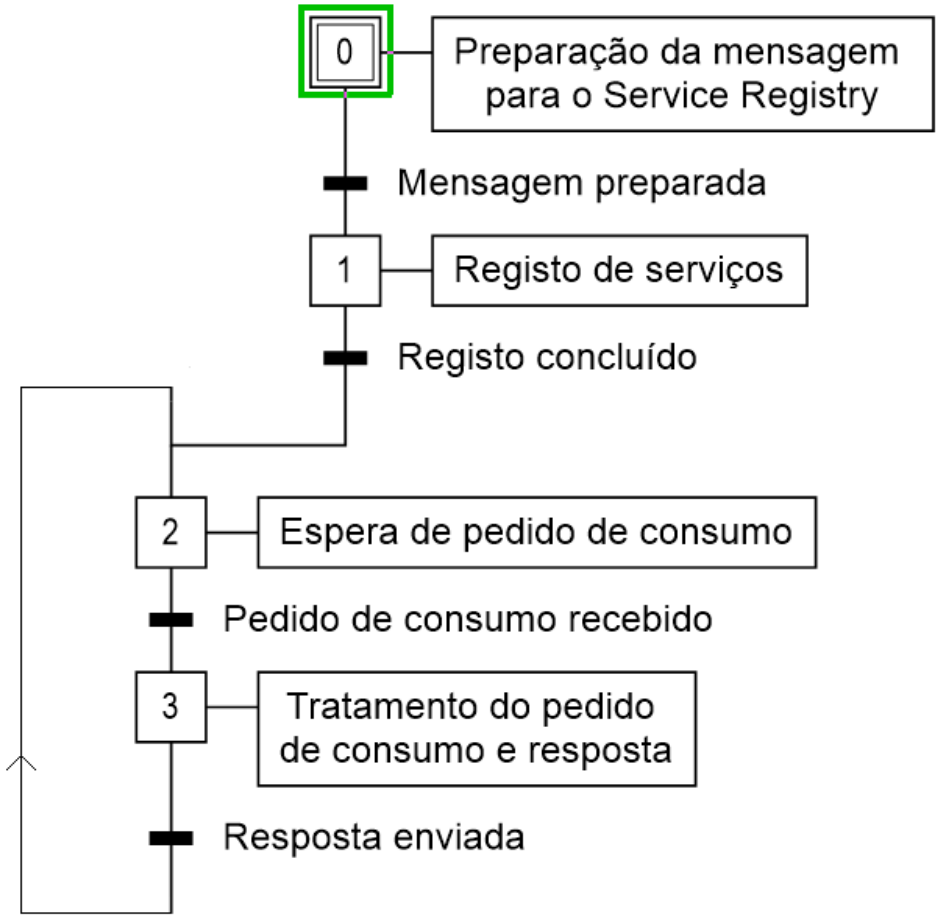

Figura 4.2 - Grafset relativo ao processo de registo e de prestação de serviço do *provider*.

Pseudocódigo exemplo do *provider*:

```
public provider = void;
main(){
        setup();
        ArrowheadFramework_handle();
}

Setup(){
        mensagem = PreparaçãomensagemJsonParaOrquestrador();
        respostaMsgOrquestrador = EnviarMsgParaOrquestrador(mensagem);
        provider = HandleResposta(respostaMsgOrquestrador);
}

ArrowheadFramework_handle(){
        Ciclo(Not end){
                If( VerificarSeExisteNovoPedido() )
                                ResponderAoPedidoDeConsumo();
        }
}
```

III - Tabela descritiva das principais funções relativas ao *consumer* da biblioteca C Arrowhead Framework.

Função	*Parâmetros de entrada*	*Parâmetros de saída*	*Descrição*
ArrowheadFramework_ConsumerInitialization	Id do *provider*; serviço; endereço; porto; unidades; diretoria url	Devolve um objeto JSON com informação sobre o *provider*	Inicializa o *consumer e efetua a orquestração*
ArrowheadFramework_Consumer_Request	Informação relativamente ao *provider* e serviço	Retorna o valor do consumo	Envia o pedido de consumo a um *provider*

IV - Tabela descritiva das principais funções relativas ao *provider* da biblioteca C *Arrowhead Framework.*

Função	*Parâmetros de entrada*	*Parâmetros de saída*	*Descrição*
ArrowheadFramework_ServiceRegister	Informação relativa ao serviço a registar	Objeto JSON relativo ao serviço registado	Regista um serviço na base de dados do Arrowhead e inicia um server
ArrowheadFramework_handle	Id do provider e valor do serviço	Retorna um código da tarefa efetuada	Trata do pedido de consumo e responde ao *consumer*
ServiceUnregister	*Objeto JSON* relativo ao serviço a eliminar	Retorna um objeto JSON com a resposta do Arrowhead	Elimina o serviço da basa de dados e termina o server

5 Extensão às IOPT-Tools

Neste capítulo é proposta uma extensão às ferramentas IOPT por forma a suportarem a geração automática de sistemas (*Application Systems*) compatíveis com o Arrowhead Framework, que podem ser *Consumers* e/ou *Providers*. Desta forma, o desenvolvimento destes sistemas é suportado por modelos gráficos que são validados e traduzidos automaticamente em código. Após a sua implementação, o *Arrowhead Framework* suporta a interação entre estes sistemas. No entanto, é necessário que os serviços do *Arrowhead framework* estejam ativos na rede comum aos *Application System*s de modo a poder usufruir dos serviços (registar e descobrir serviços).

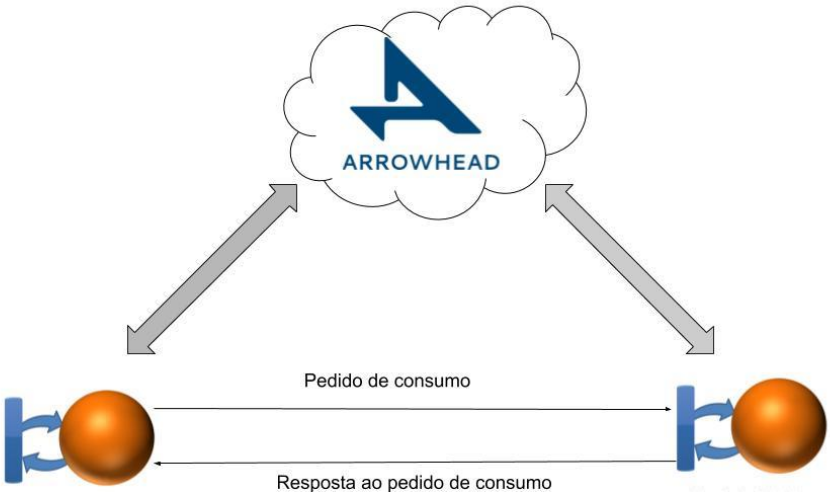

Figura 5.1 - Relação do *Arrowhead framework* com as IOPT-Tools.

É importante focar nos sinais de input/output, que servem para adquirir valores de input do hardware ou atuar em pinos de saída. Com algumas alterações nas ferramentas IOPT-Tools será possível definir os *inputs* como sinais consumidores e os *outputs* como sinais fornecedores. Para tal, é conveniente adicionar definições de *Arrowhead Framework* nas propriedades dos sinais de

input e output, a fim de tornar os sistemas desenvolvidos pelas IOPT-Tools compatíveis com o *Arrowhead Framework*.

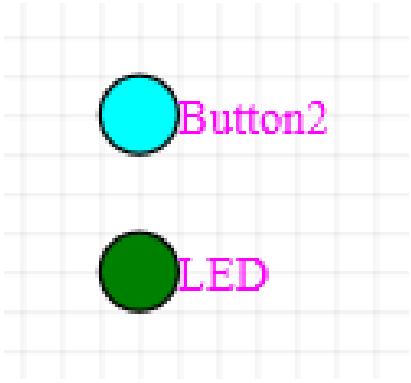

Figura 5.2 - Sinais de input e output de um modelo IOPT.

O sistema/controlador gerado pelas IOTP-Tools e executado, é cíclico passando por um momento de atualização dos sinais de input, depois é realizado o tratamento das redes de Petri para finalmente alterar os sinais de output (Figura 5.3). Quando se definir um input como *consumidor*, este terá de ser atualizado no início do ciclo, ao mesmo tempo dos restantes sinais. Do mesmo modo, para o *provider*, os sinais de output deverão ser afetados no fim do ciclo.

Assim, o consumo da medição pode ser efetuado em cada ciclo, juntamente com as leituras dos sinais de input associadas ao hardware, forçando a atualização dos sinais de input descartando por completo as medições anteriores.

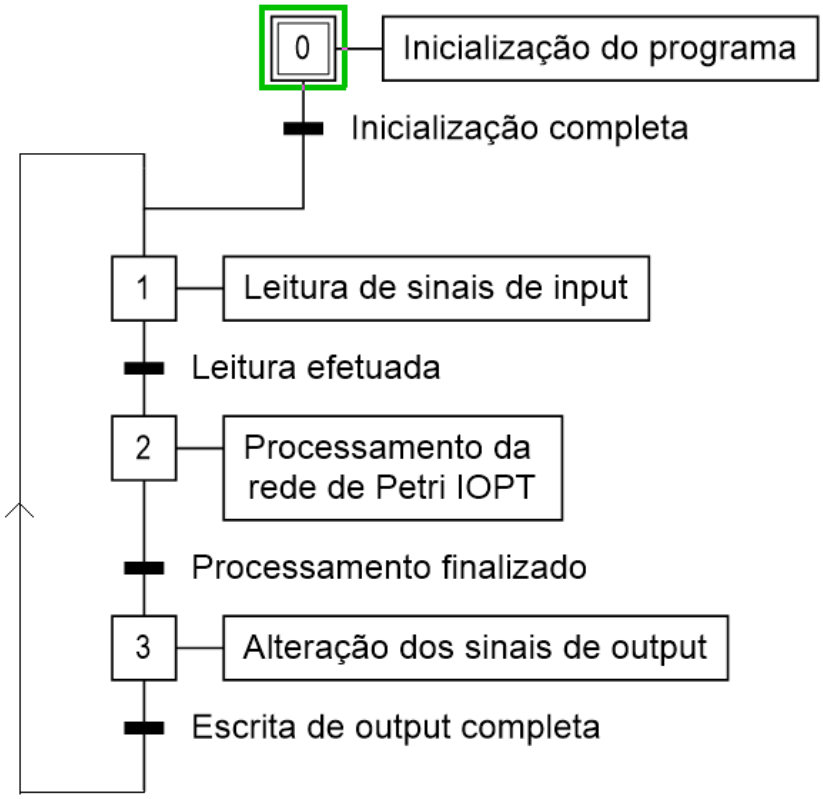

Figura 5.3 - Grafcet do funcionamento cíclico de um programa gerado pelas IOPT.

No entanto, o pedido de *orquestration* ou *discovery* pode ser realizado em várias fases do *Aplication System*. São propostas diferentes hipóteses para o problema:

1. Realizar a orquestração ou descobrimento do serviço a cada instância do ciclo, forçando atualizações constantemente, o que provoca um acréscimo no tempo necessário para ser atribuído um *provider*, e a seguir adquirir a medição por um pedido de consumo.
2. Realizar a orquestração no *setup* do programa, adquirindo e guardando o *provider* respetivo ao sinal de input, para que apenas seja necessário fazer o pedido de consumo diretamente ao *provider*, poupando o tempo da orquestração.

Ambas estas situações têm vantagens e desvantagens, principalmente devido ao tempo de processamento e espera de resposta do *orquestrador* e do *fornecedor*. Na primeira abordagem (Figura 5.4), é exigido mais tempo de processamento e recursos extra para atualizar constantemente o *provider* (realizar orquestrações sucessivas) e também os sinais de input *Arrowhead Framewor*k. Na segunda hipótese (Figura 5.5) é poupado tempo de processamento visto que o *provider* não é atualizado (apenas é realizada a orquestração uma vez). No entanto, esta abordagem traz problemas quando o *provider* deixa de estar disponível ou até mesmo quando deixa de ser o

melhor *provider* para o serviço. Esta situação é resolvida disparando um processo de orquestração periódico para atualização dos *providers*, mas também quando o *provider* deixa de responder ao pedido de consumo dentro de um intervalo de tempo definido.

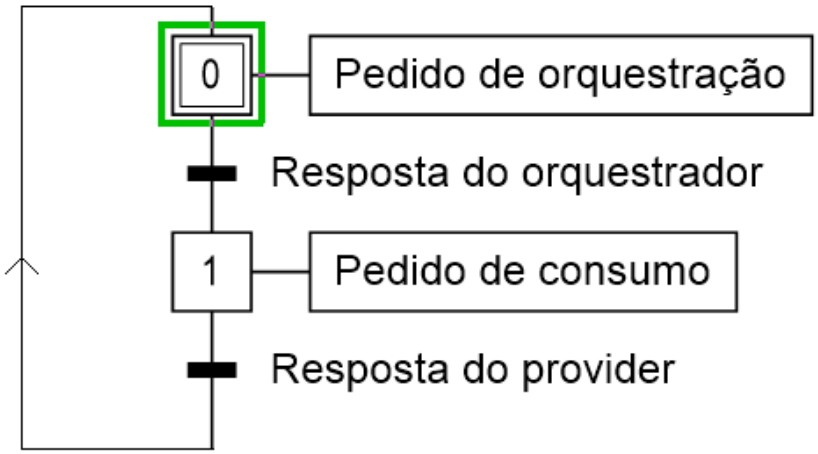

Figura 5.4 - Grafcet da hipótese 1.

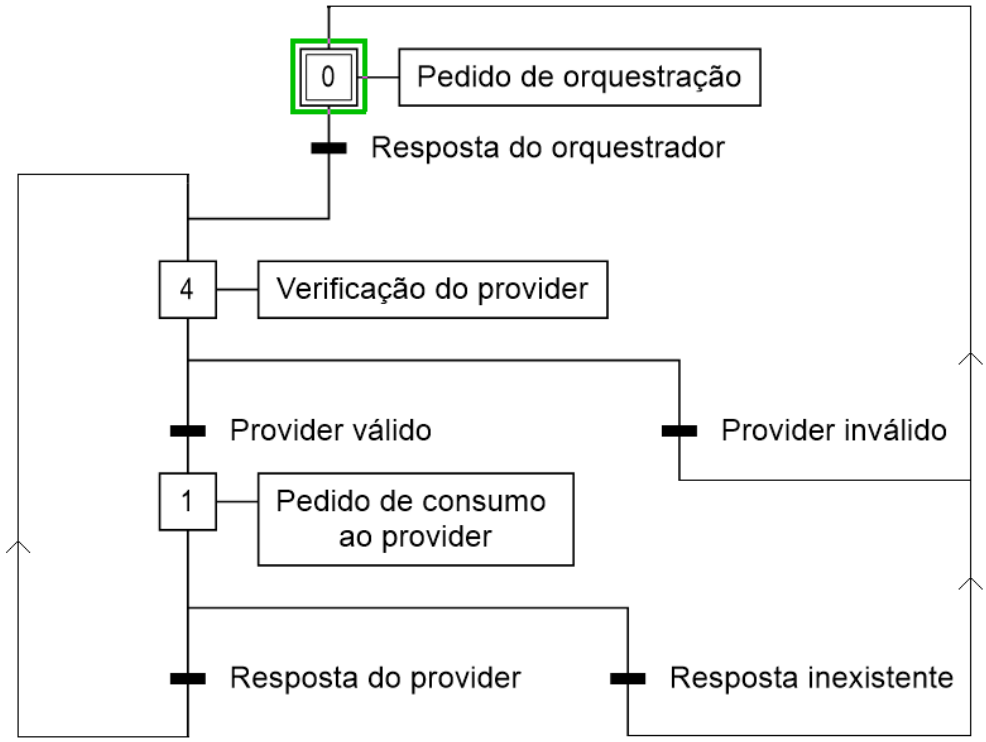

Figura 5.5 - Grafcet da hipótese 2.

Visto que tanto o processo de orquestração como o processo de consumo têm um efeito bloqueante, provocando um atraso considerável na resposta do sistema que não pode ser desprezado porque as aplicações devem ser de tempo real, é conveniente reduzir ao máximo o tempo de espera.

3. Outra abordagem é efetuar o consumo do serviço de cada sinal de entrada da rede de Petri IOPT, apenas quando uma das transições que dependem desse sinal esteja em condições para disparar (*enabled* – com os seus lugares de entrada marcados), faltando o valor desse sinal de entrada (fornecido por um *provider*) para a transição estar pronta a disparar (*ready*).

Nesta hipótese 3 (Figura 5.6) é preciso ter em conta todas as condições de cada transição para realizar o pedido de consumo, no entanto oferece melhores resultados, porque propõe-se que se faça o pedido de orquestração no *setup* da aplicação tal como na hipótese 2 (Figura 5.5), mas apenas realize o pedido de consumo ao *provider* quando essa medição for a última condição necessária para que a transição dispare. Desta forma poupa bastante tempo de processamento ao *consumer*, porque evita que esteja constantemente a comunicar com outros dispositivos. Como na hipótese 2 (Figura 5.5), existe a necessidade de atualizar o *provider* de forma periódica, tal

como em caso de demora excessiva na resposta a pedidos de consumo, deve-se assumir que o *provider* está offline e escolher novo *provider*.

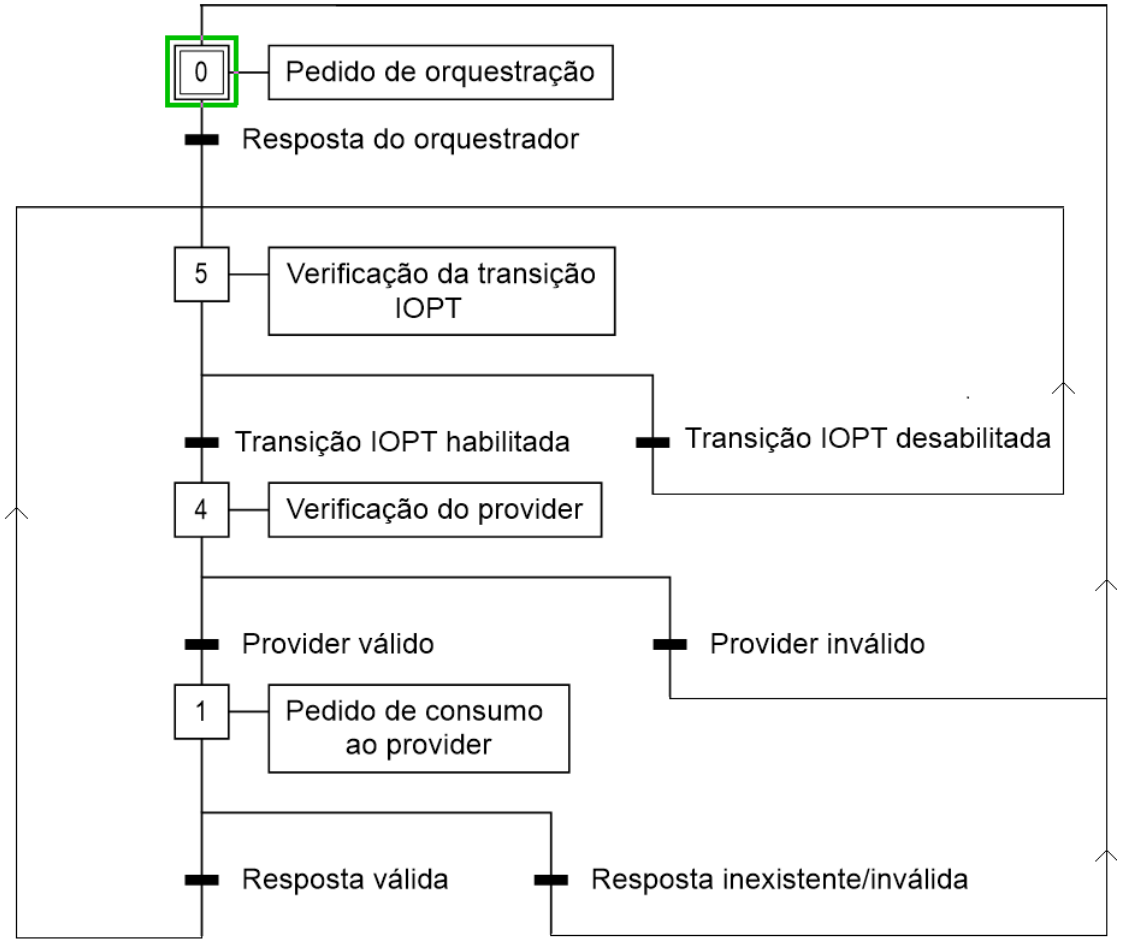

Figura 5.6 - Grafcet da hipótese 3.

Na Figura 5.7 verifica-se uma transição com um lugar de entrada (I) e outro de saída. Porque o lugar de entrada está preenchido com marcas, a transição está habilitada a disparar, faltando apenas validar a condição de guarda ArrowheadFrameworkInput ≥ 1. Nesta situação, o sinal de input do serviço de *Arrowhead Framework* (ArrowheadFrameworkInput) deve ser atualizado, para verificar a condição e disparar a transição.

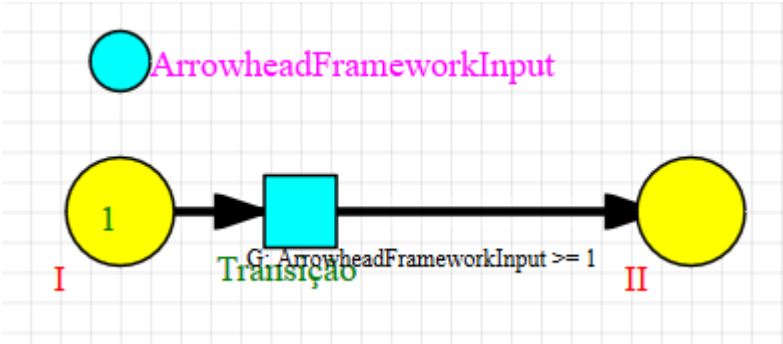

Figura 5.7 - Exemplo de uma transição habilitada.

É possível também abordar o problema numa perspetiva *publish/subscribe*. Mensagens *publish/subscribe* permitem uma comunicação assíncrona entre dispositivos baseado em serviços de eventos, estimulando performance. A fim de receber informação, o *consumer* subscreve-se ao *provider* escolhido. Assim, sempre que se verifica uma alteração das medições, o *provider* é responsável por comunicar uma publicação de modo a notificar o *consumer* dessas alterações. Esta abordagem permite reduzir o consumo de banda na rede, tempo de processamento dos dispositivos, mas também simplifica a aplicação do *consumer*.

5.1 Alterações no editor IOPT-Tools

Para adicionar os parâmetros necessários do *Arrowhead Framework* nas propriedades dos sinais de *input*, podemos inserir uma opção para que o utilizador facilmente defina qual é o modo de *output*: seja pelos pinos do dispositivo físico ou por network, via *Arrowhead Framework*.

Acionando essa escolha, fica visível um conjunto de campos obrigatórios para utilização dos serviços do *Arrowhead Framework*:

- Endereço IP do *Arrowhead Framework Core System* desejado (*Arrowhead IP*)
- O nome do serviço a consumir (*Service Name*)
- Unidade do serviço (*Service Unit*)
- Nome do *Consumer* (Consumer Name)

Quando um *Aplication System* (*consumer*) tem interesse em descobrir um serviço, é necessário conectar-se ao *Arrowhead Framework Core System*, e por isso o programador tem de definir qual é o *Arrowhead Framework* ao qual se pretende ligar, inserindo um endereço IP (na mesma rede pode existir vários *Arrowhead Frameworks*). Para alem disso, ao modelar o sistema nas IOPT-Tools, o utilizador deve definir o nome do serviço a descobrir. Deve-se também indicar quais são as unidades esperadas das medições recebidas. O nome do Cliente *Arrowhead Fra-*

mework pode corresponder ao nome do sinal de output. No entanto deve ser editável pelo programador (Id do sinal de input). A *metadata* da mensagem, informação relativamente à mensagem, como por exemplo o formato das mensagens (Figura 5.8).

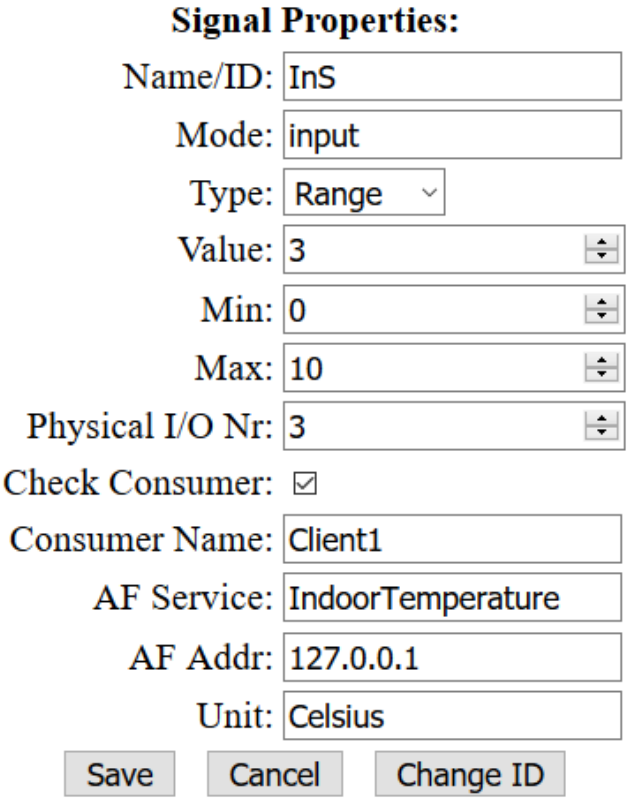

Figura 5.8 – Propriedades modificadas de um sinal de input IOPT-Tools (*consumer*).

Após a preparação da mensagem e envio, é esperado uma resposta, sendo que é devolvido informação relativamente ao *provider* ou aos *providers*, com o endereço, porto, e o caminho ("serviceURI") do serviço. Sequencialmente, o *consumer* processa a resposta do Orquestrador, quando a transição estiver em condições de disparar, a aplicação prepara o *payload* para enviar ao *provider* escolhido. Espera-se uma resposta válida do *provider* com o valor da medição, assim é associado o sinal de input à rede de Petri, atribuindo essa medição a um lugar da rede. Finalizado o processo de consumo via *Arrowhead Framework,* o programa gerado pelas IOPT-Tools continua normalmente, assumindo o valor consumido.

Numa outra perspetiva, o programa gerado pelas IOPT-Tools também pode ser um *provider*. O programa inicializa no *setup*, onde o *provider* deve registar os seus serviços ao *Service Registry*, daí em diante o *provider* deve apenas esperar pedidos de consumo diretamente vindo do *consumer*. Neste caso, o *provider* tem um comportamento de servidor, e por isso no código gerado, a aplicação do *provider* assume a forma de um servidor web (a aplicação pode ter outra abordagem). A verificação da receção de mensagens é efetuada no fim do ciclo do programa, de

modo a responder a pedidos após realizar o tratamento das redes IOPT-nets, atualizando os sinais de output.

De forma análoga ao *consumer*, para adicionar definições do *Arrowhead Framework* nas propriedades dos sinais de output, é conveniente inserir uma opção para que o utilizador facilmente defina se é para associar o sinal de output a um serviço de fornecimento do *Arrowhead Framework*, ou para atuar uma porta logica do dispositivo, ou ainda, ambas as funcionalidades (Figura 5.9).

Acionando essa escolha, fica visível um conjunto de campos obrigatório para modelar o *provider*:

- Endereço IP do *Arrowhead Framework Core System* desejado (*Arrowhead IP*)
- O nome do serviço a registar (*Service Name*)
- Porto associado (*Port*)
- Unidade do serviço (*Service Unit*)
- Nome do *provider (Provider Name)*

Para cada sinal de output que se pretende criar um serviço *Arrowhead Framework*, é necessário indicar qual o endereço do *Arrowhead Framework* de forma direcionar cada serviço para o *Arrowhead Framework* pretendido. Neste caso é necessário associar um porto para criar um http-server para aquele serviço, o nome do serviço a registar no *Service Registry*, tal como o nome do *provider* que normalmente é o nome do sinal de output (mas pode ser editado). Por fim, informação relativamente à mensagem, *metadata*.

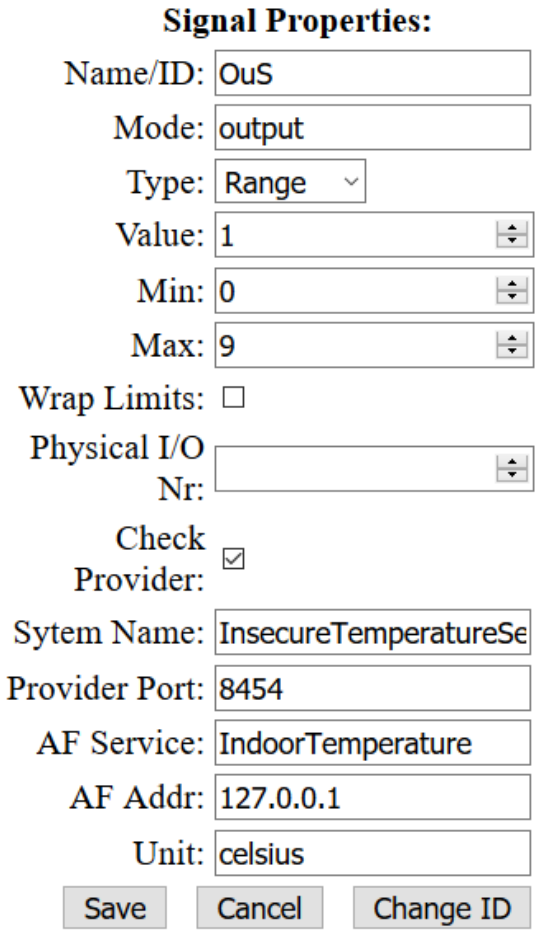

Figura 5.9 - Propriedades modificadas de um sinal de output IOPT-Tools (provider).

Visto que a *Aplication System* do *provider* e *do consumer* são independentes, o modelo pode ter em simultâneo, sinais de output e input associados a serviços de *Arrowhead Framework*, gerando um código que comunica dinamicamente com outros dispositivos, fazendo pedidos de consumo e respondendo a pedidos.

5.2 Modificações na geração de código C

Nesta secção são referidas as alterações efetuadas ao código C, automaticamente gerado pelas IOPT-Tools, de forma a que um modelo IOPT (*consumer*) possa consumir serviços de *providers* e um IOPT *provider* possa fornecer serviços a *consumers*.

Como referido na secção 2.3.2.2.4, o gerador de código C gera diversos ficheiros para compilação e execução do modelo, no entanto as alterações necessárias para que o modelo realize uma orquestração e consumo, são apenas no ficheiro net_io.c. O ficheiro net_io.c é responsável

por preparar a leitura e escrita de sinais de entrada e saída, tal como por adquirir leitura de valores analógicos dos pinos e afetar os pinos de saída com valores da rede Petri. As alterações a efetuar para um modelo *consumer* são:

1. Incluir a biblioteca "ArrowheadFramework.h" no topo do ficheiro.
2. Modificar o Makefile de modo a compilar os ficheiros do *Arrowhead framework* e utilizar o ficheiro "linux_sys_gpio" para usar pinos virtuais em Linux.
3. Na função **NomeDoModelo**_InitializeIO(), realizar a orquestração em vez de definir o modo do pino (com a função ArrowheadFramework_ConsumerInitialization() da biblioteca *ArrowheadFramework* C):
4. Na função **NomeDoModelo**_GetInputSignals,(), efetuar o consumo do serviço em vez de fazer a leitura dos pinos (com a função ArrowheadFramework_Consumer_Request() da biblioteca *ArrowheadFramework* C).

Exemplo das linhas do *consumer* a serem modificadas em net_io.c:

```
#include "ArrowheadFramework.h"
...
void Modelo_Consumer_InitializeIO()
{
    //pinMode(1, INPUT ); /* ArrowheadFramework */
    ArrowheadFramework_ConsumerInitialization("Client1","localhost","0","Indoor-
Temperature", "Celsius", "http://127.0.0.1:8440/orchestrator/orchestration", (1-1) );  /* AF
id  InS */ ...
}
void Modelo_Consumer_GetInputSignals(){
    //inputs->ArrowheadFramework = analogRead(1 ) * (10-0) / ANALOG_IN_MAX + 0;
    inputs->ArrowheadFramework = (int) ArrowheadFramework_Consumer_Request(
(1-1) ); //Arrowhead Framework Consume Request
...
}
```

De forma análoga ao *consumer*, as modificações necessárias para um modelo IOPT ser um *provider Arrowhead Framework* são apenas efetuadas em net_io.c.

1. Incluir as bibliotecas "ArrowheadFramework.h" no topo do ficheiro e criar a variável global "entry".
2. Na função **NomeDoModelo**_InitializeIO(), realizar o registo de serviços em vez de definir o modo do pino físico (com a função ArrowheadFramework_ServiceRegister() da biblioteca *Arrowhead Framework* C) e consequentemente inicializar o http-server associado ao serviço.

3. Na função **NomeDoModelo**_PutOutputSignals(), verificar a receção de pedido de consumo e responder (com a função ArrowheadFramework_handle() da biblioteca *Arrowhead Framework* C).

Exemplo das linhas do *provider* a serem modificadas em net_io.c:

```
#include "ArrowheadFramework.h"
...
void Modelo_Provider_InitializeIO()
{
      entry = ArrowheadFramework_ServiceRegister("IndoorTemperature", "cel-
sius", "InsecureTemperatureSensor", "127.0.0.1", "8454", "temperature",8454, (1-1) );
  /*OuS */
}
...
void Modelo_Provider_PutOutputSignals(){
   //pinMode( 2, OUTPUT ); /* OuS */
   if(ArrowheadFramework_handle((double)event_out->OuS, (1-1) )==2) {
     ArrowheadFramework_ServiceUnregister(entry_OuS, (1-1) );
     printf("AF SERVICE 'OuS' UNREGISTERED\n");
   }
...
}
```

5.3 Modificações efetuadas nas IOPT-Tools

Após identificar as alterações necessárias a efetuar nos ficheiros gerados pelas IOPT-Tools, alterou-se os ficheiros necessários para que as IOPT-Tools permitam a geração automática de sistemas compatíveis com o Arrowhead Framework. Foram modificados os ficheiros associados à edição dos sinais de input e output; os ficheiros de geração de código C; e foi criada uma pasta *Arrowhead Framework* com a biblioteca C (ArrowheadFramework.h e ArrowheadFramework.c) e outra biblioteca C (cJSON.h e cJSON.c) que permite o tratamento de *payload* JSON.

1. signal.html
2. pnml_editor.js
3. code_gen.php
4. iopt2c_io.xsl
5. cgMakefile

As modificações ao ficheiro signal.html permitiram incluir atributos do Arrowhead Framework no editor IOPT-Tools. Por outro lado, o ficheiro JavaScript pnml_editor.js é responsável em gerir variáveis do signal.html, dispor os respetivos atributos do Arrowhead no editor, guardar e ler dados. Foram adicionados ao ficheiro PNML gerado diversos atributos (os atributos com o prefixo "af_") associados ao Arrowhead Framework.

Exemplo do pnml de um sinal de output:

```
<signal id="OuS" type="range" value="1" gpio_nr="0" min="0" max="9"
af_addr="127.0.0.1" af_service="IndoorTemperature" wrap="0"
af_check_p="1" af_port="8454" af_unit="celsius" af_systemname="Inse-
cureTemperatureSensor">
```

Exemplo do pnml de um sinal de input:

```
<signal id="InS" type="range" value="3" gpio_nr="3" min="0" max="10"
af_addr="127.0.0.1" af_service="IndoorTemperature" af_port="8440"
af_check_c="1" af_unit="Celsius" af_clientname="Client1">
```

O ficheiro code_gen.php foi alterado para que as bibliotecas associadas ao Arrowhead Framework (Ficheiros que estão na pasta ArrowheadFramework) sejam incluídas na pasta de ficheiros gerados; iopt2c.xsl é o ficheiro que constrói o ficheiro net_io.c, sendo assim o principal alvo de modificações no que toca à geração de código C. Por fim, foi modificado o ficheiro cgMakefile de modo a compilar automaticamente o código gerado, compilar os ficheiros do Arrowhead Framework e utilizar o ficheiro “linux_sys_gpio” para usar pinos virtuais em Linux.

6 Teste e validação da contribuição

Nesta secção pretende-se testar e validar a biblioteca C desenvolvida para o *Arrohwead Framework* quanto ao seu funcionamento e eficiencia, comparando-a ao projecto java inicialmente disponibilizado pelo *Arrowhead Framework*. Pretende-se ainda testar as extensões efectuadas às IOPT-Tools. Tambem se pretende discutir os resultados da alteração do código gerado a partir de um modelo IOPT.

6.1 Biblioteca C Arrowhead Framework

Começou-se por comparar dois projectos, em C e outro em Java, quanto à sua rapidez de orquestração e consumo. Realizou-se uma extrapolação de dados recolhidos na execução dos projectos de modo a elaborar as tabelas V e VI, representando o tempo utilizado para estabelecer a ligação ao *Arrowhead Framework* e ao *provider* (realização da orquestração e pedido de consumo), para de seguida concluir diferenças entre projetos.

Cada tabela é composta por 50 amostras executadas em cada projeto, tendo extraido os seguintes resultados:

V – Tabela do consumo de tempo em segundos usados na orquestração e consumo do projeto em C.

Amostra	Tempo	Amostra	Tempo	Amos-tra	Tempo	Amos-tra	Tempo	Amostra	Tempo
1	0,700536	11	0,41318	21	0,180021	31	0,174292	41	0,19536
2	0,626746	12	0,19773	22	0,178375	32	0,193467	42	0,18517
3	0,552597	13	0,174793	23	0,347325	33	0,184253	43	0,186128
4	0,544507	14	0,178264	24	0,144843	34	0,358253	44	0,332638
5	0,444636	15	0,183507	25	0,209338	35	0,18983	45	0,165934
6	0,522897	16	0,180125	26	0,155254	36	0,163564	46	0,283165
7	0,474988	17	0,177042	27	0,256462	37	0,186797	47	0,287415
8	0,883299	18	0,183959	28	0,23259	38	0,176194	48	0,247428
9	0,585674	19	0,180682	29	0,204538	39	0,210856	49	0,235875
10	0,367215	20	0,178935	30	0,192513	40	0,224613	50	0,178288

VI – Tabela do consumo de tempo em segundos usados na orquestração e consumo do projeto Java.

Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amostra	Tempo
1	3,116	11	1,051	21	0,305	31	0,471	41	0,507
2	0,736	12	1,230	22	0,379	32	0,471	42	0,371
3	0,632	13	0,761	23	0,501	33	0,501	43	0,568
4	0,700	14	0,646	24	0,647	34	0,571	44	0,336
5	0,647	15	1,068	25	0,588	35	0,324	45	0,316
6	0,810	16	0,772	26	0,469	36	0,256	46	0,295
7	0,940	17	0,490	27	0,288	37	0,303	47	0,225
8	1,069	18	0,301	28	0,280	38	0,413	48	0,421
9	1,018	19	0,380	29	0,260	39	0,298	49	0,326
10	1,838	20	0,408	30	0,297	40	0,379	50	0,337

Com estes dados podemos anallisar com mais detalhe os resultados a partir de um grafico:

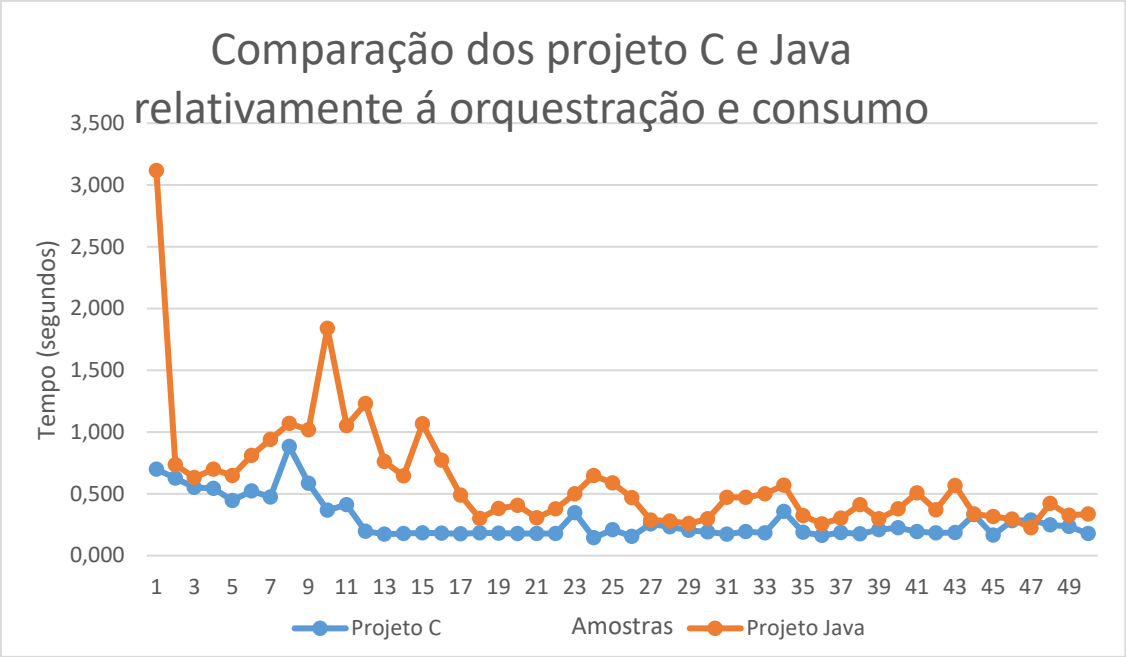

Figura 6.1 - Comparação do projeto C e Java relativamente à orquestração e consumo.

Ao observar o gráfico da Figura 6.1, verifica-se uma acentuada melhoria no tempo consumido com a biblioteca C, tanto na orquestração como no pedido de consumo, visto que a média de tempo necessário para realizar a orquestração e consumo é de 0.197 segundos com a biblioteca C e 0.646 segundos com o projeto Java. Verifica-se tambem um pico (máximo) inicial mais acentuado com o programa Java, causado pelos projetos serem corridos numa maquina virtual CentOS exigindo mais recursos e processamento, demorando mais tempo a estabilizar. Esta observação é relevante porque provavelmente a *Application system* apenas necessita realizar uma orquestração periodicamente e por isso o projeto C para além de ser mais rápido em média, anuncia-se mais eficiente. Este aspecto é importante para que os sistemas desenvolvidos com as IOPT-Tols comuniquem dentro de um intervalo de tempo máximo, criando sistemas de tempo real.

Agora podemos fazer uma análise de tempo do pedido de consumo, ou seja, depois de realizar a orquestração tendo um *provider* disponivel, podemos colocar o *consumer* a fazer sequencialmente vários pedidos de consumo de modo a tabular o tempo de resposta do processo:

VII - Tabela do consumo de tempo em segundos usados no pedido de consumo do projeto C.

Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amostra	Tempo
1	0,035	11	0,040	21	0,042	31	0,038	41	0,034
2	0,041	12	0,037	22	0,031	32	0,032	42	0,034
3	0,045	13	0,028	23	0,027	33	0,037	43	0,030
4	0,046	14	0,040	24	0,030	34	0,033	44	0,031
5	0,045	15	0,032	25	0,035	35	0,040	45	0,036
6	0,043	16	0,024	26	0,035	36	0,035	46	0,032
7	0,046	17	0,035	27	0,037	37	0,038	47	0,033
8	0,035	18	0,032	28	0,025	38	0,026	48	0,027
9	0,033	19	0,032	29	0,030	39	0,033	49	0,029
10	0,055	20	0,037	30	0,033	40	0,026	50	0,036

VIII - Tabela do consumo de tempo em segundos usados no pedido de consumo do projeto Java.

Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amostra	Tempo	Amos-tra	Tempo
1	0,396	11	0,194	21	0,104	31	0,078	41	0,049
2	0,111	12	0,111	22	0,053	32	0,081	42	0,057
3	0,189	13	0,152	23	0,107	33	0,069	43	0,080
4	0,120	14	0,134	24	0,067	34	0,055	44	0,081
5	0,190	15	0,177	25	0,130	35	0,069	45	0,063
6	0,179	16	0,143	26	0,124	36	0,088	46	0,066
7	0,225	17	0,125	27	0,062	37	0,046	47	0,075
8	0,134	18	0,081	28	0,093	38	0,072	48	0,130
9	0,087	19	0,095	29	0,083	39	0,045	49	0,076
10	0,211	20	0,089	30	0,097	40	0,162	50	0,117

No gráfico da Figura 6.2, conseguimos provar a fiabilidade e regularidade do projeto em C no consumo de serviços ao *provider*, tendo em conta que na *Aplication System* é bastante provável o múltiplo consumo sucessivo do mesmo serviço (sem realizar nova *orquestration*), então é conveniente para bom funcionamento do programa em tempo real que os dispositivos sejam o mais rápido e regular possível na comunicação. O projeto C garante ser mais de 3 vezes mais rápido no consumo de um serviço do que o projeto Java com uma média de 0.034 segundos e 0.112 segundos para o projeto Java.

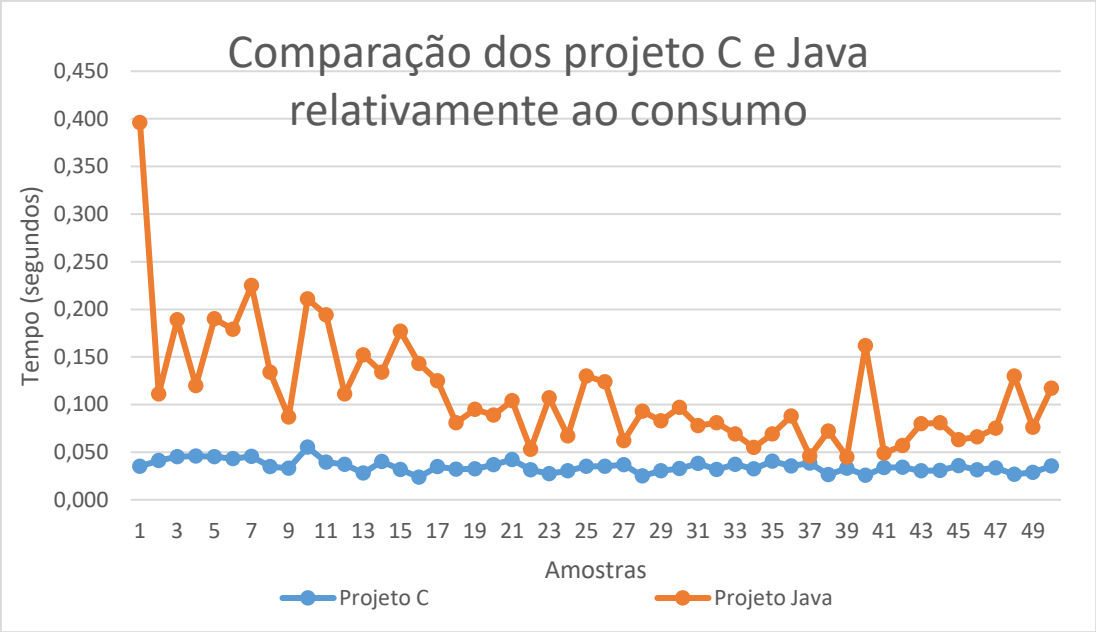

Figura 6.2 - Comparação dos projetos C e Java relativamente ao consumo.

Por fim, ao comparar os tempos da tabela VII do projeto C (orquestração e consumo) com a tabela VIII (consumo), resulta no grafico da Figura 6.3, onde se verifica que o tempo médio do processo de consumo é cerca de 6 vezes menor do que realizar a orquestração e o consumo. Esta observação mostra que é muito conveniente evitar multiplas orquestraçoes poupando tempo e recursos da rede. Estes resultados são causados pelo *Arrowhead Framework orquestration core system* que consome tempo na comunicação com o respetivo Service Registry core system, e na iteração dos serviços disponiveis na escolha do melhor *provider*.

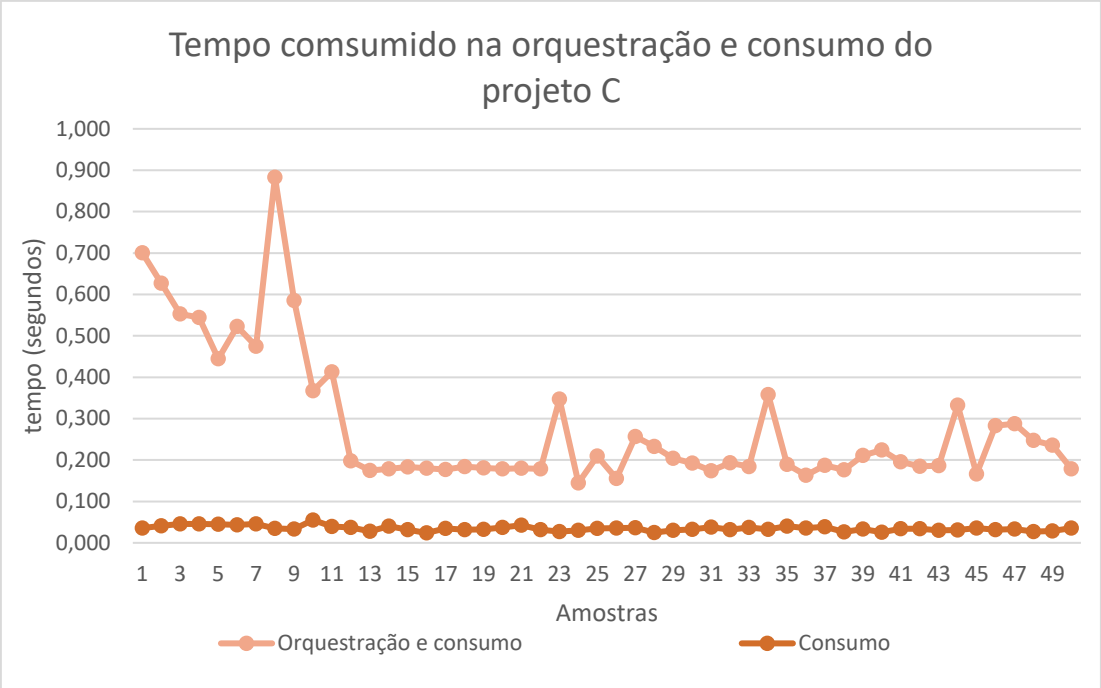

Figura 6.3 - Tempo consumido na orquestração e pedido de consumo.

É de ter em conta que ambos estes processos efetuados são realizados num computador pessoal e por isso estes dados podem variar consoante a capacidade de processamento da máquina utilizada, mas também consoante à qualidade da ligação de rede entre consumer, *Arrowhead framework* cores e provider.

6.2 Validação das IOPT-Tools

6.2.1 Teste com modificações manuais no código gerado

De modo a validar as modificações manuais do código gerado a partir de um modelo, usou-se um conjunto de *consumers* gerados pelas IOPT-Tools.

A realização destes testes baseou-se na execução de *providers* que fornecem dados a *consumers* IOPT. Nesta primeira fase, o *provider* é um programa teste baseado na biblioteca C desenvolvida. Inicialmente o *consumer* realiza a orquestração para escolher um *provider* do serviço, feito isso, o programa realiza constantemente: um pedido de consumo recebendo resposta e atualizando o sinal de input, o tratamento da rede de Petri IOPT-Tools e por fim afeta os sinais de output.

Tendo realizado com sucesso testes aos *consumers*, realizaram-se também testes a *providers* gerados pelas IOPT-Tools. Após preparar um *consumer* e um *provider* IOPT, executaram-

se as aplicações no terminal. Foi possível visualizar as mudanças de estados de cada programa, verificando também os pedidos de consumo e respostas associadas.

Os programas manualmente alterados corresponderam às espectativas, por isso foi possível prosseguir para a alterações no gerador de código das IOPT-Tools, de modo a gerar automaticamente *consumer* e *providers* sem existir a necessidade de alterações manuais por parte do programador.

6.2.2 Teste de *consumer* e *provider* gerados pelas IOPT-Tools

Para poder testar a comunicação de um *consumer* com um *provider* gerado automaticamente pelas IOPT-Tools, modelou-se um *consumer* que depende da leitura de um serviço de modo a habilitar a transição IOPT-net e modelou-se também um *provider* que disponibiliza esse serviço. Executaram-se os dois programas gerados e verificou-se que o *consumer* recebe informação e consequentemente altera o seu estado.

Verificando que as IOPT-Tools conseguem gerar um *consumer* e um *provider*, e que a execução do programa é bem-sucedida, então falta verificar o correto funcionamento quando são criados vários *consumers* e *providers* no mesmo modelo IOPT. Para isso foram modelados sistemas com vários *consumers* e sistemas com vários *providers*. O objetivo principal foi verificar que um controlador permite o consumo de vários serviços, e também por outro lado, outro controlador ser capar de fornecer serviços distintos corretamente.

Ambos os testes efetuados foram bem-sucedidos. Assim, como esperado as IOPT-Tools permitem a geração de diversos serviços de consumo ou de fornecimento.

7 Conclusões e Trabalhos Futuros

Nesta dissertação realizou-se um estudo aprofundado das ferramentas IOPT-Tools e *Arrowhead Framework* e foram elaboradas um conjunto de hipóteses por forma a tornar estas ferramentas compatíveis. O trabalho desenvolvido resultou numa publicação de um artigo científico, nas XIV Jornadas sobre Sistemas Reconfiguráveis (REC 2018), com o título "Componente modular reconfigurável para indústria 4.0" [44].

Foram instaladas e executadas as ferramentas base do Arrowhead Framework de modo a entender o seu funcionamento numa perspetiva prática. Dessa análise, verificou-se a necessidade de criar uma biblioteca C com as funcionalidades do Arrowhead Framework de forma a poder mais facilmente adaptar as IOPT-Tools. Com isto elaborou-se algumas hipóteses compatíveis com as ferramentas e os objetivos definidos no início do projeto. De seguida identificaram-se as alterações necessárias a fazer no código gerado pelas IOPT-Tools e aplicaram-se essas alterações manualmente de forma a poder testar e validar as hipóteses. Depois foram efetuadas modificações no editor das IOPT-Tools de modo a receber parametrizações dos sinais ligados ao *Arrowhead Framework*. Finalmente, aplicou-se a biblioteca C de *Arrowhead Framework* ao gerador automático de código C das IOPT-Tools. Estas alterações no gerador de código permitiram finalizar os objetivos definidos no início da dissertação e realizar testes de geração automática de código de modelos ligados ao *Arrowhead Framework*. Este fluxo de desenvolvimento permitiu comprovar a adequabilidade das IOPT-Tools para o desenvolvimento de sistemas compatíveis com o Arrowhead Framework.

Relativamente à biblioteca C criada, realizaram-se testes de forma a provar a eficiência e regularidade comparativamente ao projeto Java. Também se provou que o processo de orquestração é mais demorado e por isso é conveniente reduzir a frequência de orquestração. Adicionalmente, a biblioteca C mostrou-se muito mais leve no consumo de recursos, permitindo o uso das funcionalidades do *Arrowhead Framework* em hardware com menos capacidade de processamento e de recursos.

Nos testes efetuados, verificou-se que os consumers tendem estar constantemente a fazer pedidos de consumo, por isso com uma abordagem publish/subscribe o fluxo de mensagens pode ser reduzido.

Durante o desenvolvimento do projeto surgiram algumas adversidades, nomeadamente na instalação e execução do *Arrowhead Framework*, devido a abundância de informação teórica sobre o assunto, no entanto havendo falta de objetividade a nível prático. Também surgiram adversidades no desenvolvimento da biblioteca C do *Arrowhead Framework* provocada pela redução de nível de abstração de programação (Java para C).

Concluindo, esta dissertação apresenta uma proposta que possibilita as IOPT-Tools desenvolver automaticamente sistemas/controladores que se conectem dinamicamente com outros dispositivos desconhecidos, permitindo realizar uma extração de dados de sistemas alheiros ao sistema. Esta proposta permite criar modelos flexíveis e dinâmicos, e também oferece novas soluções para projetos futuros.

Ao trabalho realizado ainda é possível adicionar diversas melhorias e novas implementações que sirvam de mais valias ao projeto. Entre as quais sugerem-se:

1. Permitir que o *provider* possa registar vários serviços diferentes. Para isso é necessário modificar o gerador automático de código C das IOPT-Tools de forma a admitir vários registos de serviços diferentes.
2. Aplicar um método Publish/Subcrive na geração de modelos *providers/consumers* IOPT-Tools. Como concluído, uma abordagem publish/*subscribe* pode ser conveniente, mas para isso é necessário modificar o editor de forma a dar escolha ao programador sobre o método de comunicação, depois é necessário alterar a biblioteca C implementando essa comunicação.
3. Aplicação desta hipótese nas IOPT-Flow. A partir deste projeto, pode-se aplicar estas hipóteses e métodos a outras ferramentas.

Referências

[1] P. Borrel, “Un monde sans travail ?,” RTP, France, 2017.

[2] J. M. Tien, “The next industrial revolution: Integrated services and goods,” *J. Syst. Sci. Syst. Eng.*, 2012.

[3] V. Roblek, M. Meško, and A. Krapež, “A Complex View of Industry 4.0,” *SAGE Open*, vol. 6, no. 2, 2016.

[4] T. Wagner, C. Herrmann, and S. Thiede, “Industry 4.0 Impacts on Lean Production Systems,” in *Procedia CIRP*, 2017.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[6] “Arrowhead | Ahead of the future.” [Online]. Available: http://www.arrowhead.eu/. [Accessed: 08-Dec-2017].

[7] “Productive 4.0 - A European co-funded innovation and lighthouse project on Digital Industry.” [Online]. Available: https://productive40.eu/. [Accessed: 03-Sep-2018].

[8] M. Boshernitsan and M. Downes, “Visual Programming Languages: A Survey,” no. Report No. UCB/CSD-04-1368, 1997.

[9] F. Moutinho, F. Pereira, and L. Gomes, “IOPT Tools User Manual,” vol. 2014, no. C, pp. 1–50, 2014.

[10] “What Is Industry 4.0, Anyway? > ENGINEERING.com.” [Online]. Available: https://www.engineering.com/AdvancedManufacturing/ArticleID/16521/What-Is-Industry-40-Anyway.aspx. [Accessed: 30-Aug-2018].

[11] “Industry 4.0 - the Nine Technologies Transforming Industrial Production.” [Online]. Available: https://www.bcg.com/capabilities/operations/embracing-industry-4.0-rediscovering-growth.aspx. [Accessed: 30-Aug-2018].

[12] K. Park, M. C. Nguyen, and H. Won, “Web-based collaborative big data analytics on big data as a service platform,” *Int. Conf. Adv. Commun. Technol. ICACT*, vol. 2015–Augus, pp. 564–567, 2015.

[13] “C-Thru - Smoke Diving Helmet on Vimeo.” [Online]. Available: https://vimeo.com/173241429. [Accessed: 03-Mar-2018].

[14] “Arena Simulation.” [Online]. Available: https://www.arenasimulation.com/. [Accessed: 03-Mar-2018].

[15] “Flexsim Express - Software de simulação grátis.” [Online]. Available: https://www.flexsim.com/pt/flexsim-express/. [Accessed: 03-Mar-2018].

[16] L. Bassi, “Industry 4.0: Hope, hype or revolution?,” *RTSI 2017 - IEEE 3rd Int. Forum Res. Technol. Soc. Ind. Conf. Proc.*, 2017.

[17] F. Ppp, “Factories 4.0 and Beyond,” 2016.

[18] L. Spendla, M. Kebisek, P. Tanuska, and L. Hrcka, “Concept of predictive maintenance of production systems in accordance with industry 4.0,” *SAMI 2017 - IEEE 15th Int. Symp. Appl. Mach. Intell. Informatics, Proc.*, pp. 405–410, 2017.

[19] E. Eronu, S. Misra, and M. Aibinu, “Reconfiguration approaches in Wireless Sensor Network: Issues and challenges,” *2nd Int. Conf. Emerg. Sustain. Technol. Power ICT a Dev. Soc. IEEE NIGERCON 2013 - Proc.*, pp. 143–152, 2013.

[20] J. P. Diprose, “End user robot programming via visual languages,” *2011 IEEE Symp. Vis. Lang. Human-Centric Comput.*, no. Figure 1, pp. 229–230, 2011.

[21] B. J. Smith, “Conceptual graphs as a visual programming language for teaching programming,” *Vis. Lang. Human-Centric Comput. 2009. VL/HCC 2009. IEEE Symp.*, pp. 258–259, 2009.

[22] “Node-RED.” [Online]. Available: https://nodered.org/. [Accessed: 27-Aug-2018].

[23] A. Rajalakshmi and H. Shahnasser, “Internet of Things using Node-Red and alexa,” *2017 17th Int. Symp. Commun. Inf. Technol.*, pp. 1–4, 2017.

[24] Z. Chaczko and R. Braun, “Learning data engineering: Creating IoT apps using the node-RED and the RPI technologies,” *2017 16th Int. Conf. Inf. Technol. Based High. Educ. Training, ITHET 2017*, 2017.

[25] “NTK | NETLabTK: Tools for Tangible Design.” [Online]. Available: http://www.netlabtoolkit.org/. [Accessed: 27-Aug-2018].

[26] “Ardublock | A Graphical Programming Language for Arduino.” [Online]. Available: http://blog.ardublock.com/. [Accessed: 28-Aug-2018].

[27] A. B. Pratomo and R. S. Perdana, “Arduviz, a visual programming IDE for arduino,” *Proc. 2017 Int. Conf. Data Softw. Eng. ICoDSE 2017*, vol. 2018–Janua, pp. 1–6, 2018.

[28] P. P. Ray, “A Survey on Visual Programming Languages in,” vol. 2017, 2017.

[29] “Get started - AT&T Flow.” [Online]. Available: https://flow.att.com/start. [Accessed: 28-Aug-2018].

[30] C. Environment and C. D. Face, “AT & T Flow Designer,” pp. 1–5.

[31] “Reactive Blocks | Bitreactive.” [Online]. Available: http://www.bitreactive.com/reactive-blocks/. [Accessed: 28-Aug-2018].

[32] A. Kraemer, “Block by Block Towards IoT Applications.”

[33] “MATLAB - MathWorks - MATLAB & Simulink.” [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed: 28-Aug-2018].

[34] M. Foltin, “ThingSpeak - IoT Platform with MATLAB Analytics.”

[35] L. Gomes, F. Moutinho, F. Pereira, J. Ribeiro, A. Costa, and J. P. Barros, “Extending input-output place-transition Petri nets for distributed controller systems development,” *Proc. - 2014 Int. Conf. Mechatronics Control. ICMC 2014*, no. Icmc, pp. 1099–1104,

2015.

[36] T. Murata, “Petri Nets : Properties , Analysis and Appl k a t ions,” vol. 77, no. 4, pp. 541–580, 1989.

[37] F. Pereira, F. Moutinho, and L. Gomes, “IOPT-tools-Towards cloud design automation of digital controllers with Petri nets,” *Proc. - 2014 Int. Conf. Mechatronics Control. ICMC 2014*, no. Icmc, pp. 2414–2419, 2015.

[38] R. Campos-Rebelo, F. Pereira, F. Moutinho, and L. Gomes, “From IOPT Petri nets to C: An automatic code generator tool,” *IEEE Int. Conf. Ind. Informatics*, pp. 390–395, 2011.

[39] F. Pereira and L. Gomes, “Automatic synthesis of VHDL hardware components from IOPT Petri net models,” *IECON Proc. (Industrial Electron. Conf.*, pp. 2214–2219, 2013.

[40] F. Joaquim and G. Pereira, “The DS-Pnet modeling formalism for cyber-physical system development,” 2017.

[41] F. Pereira and L. Gomes, “The IOPT-Flow framework pairing Petri nets and data-flows for embedded controller development,” *IECON Proc. (Industrial Electron. Conf.*, pp. 4832–4837, 2016.

[42] F. Moutinho, L. Paiva, J. Kopke, and P. Malo, “Extended Semantic Annotations for Generating Translators in the Arrowhead Framework,” *IEEE Trans. Ind. Informatics*, vol. 3203, no. c, pp. 1–1, 2017.

[43] H. Derhamy, J. Ronnholm, J. Delsing, J. Eliasson, and J. van Deventer, “Protocol interoperability of OPC UA in service oriented architectures,” *2017 IEEE 15th Int. Conf. Ind. Informatics*, pp. 44–50, 2017.

[44] T. P. Noronha, P. Rodrigues, F. Moutinho, R. Campos-Rebelo, P. Maló, and L. Gomes, “Componente modular reconfigurável para a indústria 4.0,” *REC*, 2018 - XIV Jornadas sobre Sistemas Reconfiguráveis, Portugal, 2018.