INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Desenvolvimento de um Sistema de Transporte Modular com Auto-Organização de Componentes Ciber-Físicos

Joy Jaques Ferreira Teixeira

Dissertação apresentada à
Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Bragança

para obtenção do grau de Mestre em

Engenharia Industrial

ramo de especialização em

Engenharia Eletrotécnica

Este trabalho foi efetuado sob orientação de:

Prof. Dr. Paulo Jorge Pinto Leitão

Prof. Dr. José Fernando Lopes Barbosa

2015/2016

Agradecimentos

A realização deste relatório de projeto marca o fim de uma importante etapa da minha vida. É com muita satisfação que expresso aqui o mais profundo agradecimento a todos aqueles que tornaram a realização deste trabalho concretizado.

Em primeiro lugar gostaria de agradecer aos professores Doutores Paulo Leitão e José Barbosa por toda a sua dedicação, motivação, conhecimento, e pela enorme disponibilidade em estar presente para a realização desta tese, sem o qual a conclusão não seria possível.

Gostaria também de agradecer a todos presentes no laboratório de Engenharia Eletrotécnica, pela contribuição, acolhimento, apoio e amizade prestada durante a elaboração do projeto.

Não menos importante queria agradecer aos meus Pais e a minha irmã, e a minha família, pelo seu incansável apoio em todo o percurso académico e apesar de todos os obstáculos, eles estiveram sempre presentes ao meu lado para chegar até esta fase final.

Um agradecimento muito especial a minha namorada que me tem apoiado ao longo deste percurso e que tem sido uma grande ajuda em todo o desenrolar desta minha vida académica.

Por ultimo, não menos importante, quero agradecer a todos os meus amigos, pelo apoio, incentivo, motivação e companheirismo ao longo deste percurso académico.

Resumo

O relatório de projeto descreve o desenvolvimento de um sistema de transporte modular auto-organizado de componentes ciber-físicos. Este sistema é constituído por componentes ciber-físicos, sendo que a parte física é constituída por uma correia transportadora, operando com um motor DC 24V e com sensores fotoelétricos de entrada e saída, enquanto que a parte lógica é processada no Raspberry Pi, consistindo num minicomputador onde os agentes são implementados.

Acoplaram-se vários componentes ciber-físicos, permitindo transportar peças do ponto A ao ponto B, desenvolvendo um sistema transportador de peças com mecanismo de auto-organização através da criação de um sistema multi-agente, com paradigmas distribuídos e inteligentes.

A programação dos agentes foi desenvolvida em linguagem Java na plataforma JADE - JAVA Agente Development Framework. O desenvolvimento dos agentes consistiu em criar um mecanismo de auto-organização através de regras simples, que permite o funcionamento do sistema dinamicamente, incluindo, como propriedades a possibilidade de adicionar e/ou remover transportadores, ou mesma a alteração da sequência dos transportadores, mantendo a operacionalidade e dinamismo do sistema automaticamente sem a necessidade de parar, reprogramar ou mesmo reiniciar o sistema.

Palavras-chave: Auto-organização, sistema Ciber-Fisicos, sistema multi-agente.

Abstract

The project report describes the development of a modular and self-organized transport system of cyber-physical components. This system consists of cyber-physical components, where the physical part consists of a conveyor belt, operating with a DC 24V motor and with photoelectric input and output sensors. The logic part is processed in a Raspberry Pi, consisting of a Minicomputer where the agents are implemented.

Several cyber-physical components may be coupled to carry parts from point A to point B. The development of the conveyor system comprises self-organizing mechanisms through the creation of a multi-agent system with distributed and intelligent paradigms.

The agent development was performed in the Java programming language On the JADE platform - JAVA Agent Development Framework. The development of the agents consisted of creating a mechanism of self-organization through simple rules, which allows the system to operate dynamically, including, as properties, the possibility of adding and / or removing conveyors, or even on changing the sequence of the conveyors. Lastly, the self-organization mechanisms keep the functionality and dynamism of the system automatically without the need to stop, reprogram or even restart the system.

Keywords: Self-organization, Cyber-Physical Systems, Multi-Agent System

Índice

Lista de Figuras

Lista de Abreviaturas

ACC - Agent Communication Chanel

ACL - Agent Communication Language

AMS - Agent Management System

AWT - Abstract Windowing Toolkit

CPC - Cyber-Physical-Component

CPS – Cyber physical System

DF - Directory Facilitator

FIPA - Foundation for Intelligent Physical Agent

GPIO – General Purpose Input/output

JADE - JAVA Agent Development Framework

JRE - Java Runtime Environment

JVM - Java Virtual Machine

MAS – Multi-Agent System

PLC - Controlador Lógico Programável

SMA - Sistema Multi-Agent

Capítulo1

1. Introdução

A manufatura é fundamentalmente uma força que permanece para conduzir o mundo a um crescimento económico [ElMaraghy, 2006]. Em particular, o setor da indústria emprega 31 milhões de pessoas na união europeia em 2009, e tem gerado 5812 biliões EUR em negócios. Nos últimos anos, as empresas de produção industrial estão a enfrentar fortes pressões com o custo, qualidade e personalização de produtos [ElMaraghy, 2006], o que exige sistemas reconfiguráveis e sistemas com capacidade de resposta a manter níveis de competitividade.

A importância da utilização dos princípios da Industria 4.0 [Kagermann et al., 2013], conhecida como a quarta revolução industrial, é caracterizada como fábricas inteligentes, com estruturas modulares, sendo que os sistemas ciber-físicos monitorizam os processos físicos, tomando decisões descentralizadas. Com a internet das coisas, os sistemas ciber-físicos comunicam e cooperam entre si e com os humanos em tempo real, através da computação em nuvem. Uma observação importante é que esta revolução traz, não só benefícios, mas também, 40% a 50% de substituição e/ou adaptação de equipamentos [Bauer et al., 2015].

Os CPS (Cyber-Physical System) podem ser vistos como um modo de transformar as fábricas tradicionais em fábricas inteligentes da Industrias 4.0 [ACATECH, 2011]. Os sistemas multi-agentes (MAS) [Leitão, 2009b; Ferber, 1999] são uma tecnologia para implementar soluções CPS, complementado e combinado com outras tecnologias, tais como, computação em nuvem e comunicação Machine-to-Machine (M2M). Os MAS permitem a resolução de problemas da engenharia, utilizando funções de controlo, software de agentes inteligentes, que cooperam em conjunto para alcançar os objetivos do sistema.

Este tipo de sistemas tem de ser modular, robusto e escalável, mas também estar ciente da reconfiguração, adaptação e evolução de uma forma muito rápida, automática e auto-forma. Este problema pode ser resolvido pela utilização da inspiração biológica que permite tomar algumas decisões simples de mecanismos poderosos que estão a trabalhar na natureza e ser traduzido para o domínio da engenharia para melhorar estes sistemas a serem mais respondíveis e ágeis para emergência.

A auto-organização é um processo de evolução, onde os desenvolvimentos de estruturas complexas levam a exibirem comportamentos que não são facilmente

previsíveis tendo conhecimento apenas das interações entre os constituintes desse sistema. A auto-organização pode caracterizar-se como a capacidade de um sistema criar padrões de comportamentos não previsíveis e descentralizados.

Com isto, o trabalho desenvolvido consiste no uso de um sistema de componentes ciber-físicos e desenvolver um sistema em que cada transportador é capaz de comunicar um com os outros e cada um deles tem a propriedade de tomar a sua própria decisão e principalmente ser auto-organizado, reconhecendo a sua posição atual, ou reconhecer a sua nova posição ou mesmo a sua remoção no sistema. Para isso, como hardware foi utilizado um Raspberry Pi para cada transportador, contendo a programação dos agentes desenvolvida no Software Eclipse.

1.1 Objetivos e motivação

O objetivo deste trabalho é desenvolver um sistema de transporte modular auto-organizado. Para isso, irá ser usada a tecnologia multi-agentes como camada lógica, permitindo que o desenvolvimento do sistema CPS seja modular e distribuído. Para isso é necessário estudar e explorar a criação de CPS auto-organizados com um conjunto de regras impostas de modo a que os agentes do sistema se comuniquem entres eles e que tornam o sistema multi-agentes automatizado e autónomo. Tendo isto em mente, o objetivo deste relatório de projeto é discutir os benefícios dos sistemas baseados em agentes com a capacidade de auto-organização e, particularmente, para mostrar a implementação de CPS auto-organizados por um sistema de módulos transportadores, onde a configuração dinâmica do sistema permite automaticamente reconfigurar o sistema, sem a necessidade de parar, reprogramar ou mesmo reiniciar o sistema.

1.2 Organização do trabalho

Este trabalho é constituído por 6 capítulos. Após a presente introdução, segue-se a descrição dos agentes e sistema multi-agentes, a sua definição, as suas propriedades, a sua implementação, o desenvolvimento do sistema multi-agentes, os diversos softwares utilizados para a criação do MAS e a linguagem de comunicação.

No terceiro capítulo é descrito o caso de estudo, quais os componentes a ser utilizados no sistema e como irá ser desenvolvido o sistema modular do transportadores auto-organizados.

O quarto capítulo refere-se à engenharia do sistema desenvolvido neste projeto, baseado em CPS auto-organizados, descrevendo o(s) hardware(s) utilizados, o funcionamento do sistema em situação inicial e normal e por fim e não menos importante, é descrito o desenvolvimento dos agentes implementados no Raspberry Pi.

O quinto capítulo refere-se ao mecanismo de auto-organização utilizado neste relatório de projeto, o plug-in e plug-out de transportadores, a alteração da sequência dos transportadores para assim concretizar os objetivos inicias.

Por fim são apresentadas as conclusões relativas ao relatório de projeto.

Capítulo 2

2. Agentes e sistemas multi-agentes

No decorrer dos últimos anos verificou-se um aumento na investigação relativamente a área de agentes inteligentes ou autónomos. Esta área de investigação surgiu inspiradas nas áreas cientificas da Inteligência Artificial, Sistemas Distribuídos, Engenharia de Software e Redes de Computadores, Sociologia, Teoria de Jogos e Economia. [ACATECH,2011]

- **Inteligência Artificial** – Micro-aspetos, como a resolução de problemas, raciocínio lógico, representação e utilização de conhecimento, planeamento, aprendizagem, etc.;
- **Engenharia de Software** – O agente como uma abstração, programação orientada por agentes;
- **Sistemas Distribuídos e Redes de Computadores** – Arquiteturas de agentes, Sistemas Multi-Agente, comunicação e coordenação;
- **Sociologia** – Macro-aspetos como a formação de sociedades virtuais e a interação entre agentes;
- **Teoria dos Jogos e Economia** – Negociação, resolução de conflitos, mecanismos de mercado.

A inteligência artificial teve inicialmente uma influência bastante elevada sobre os campos dos agentes autónomos e sistemas multi-agentes, hoje em dia, este campo evoluiu muito para além de já ser considerado uma subárea da inteligência artificial.

2.1 Definição de Agente

Um agente é um sistema computacional situado num determinado ambiente. Este ambiente pode fazer parte de um mundo real (universidade, hospital, industria, campo de futebol, etc.), ou mesmo, num ambiente de simulação (computador). Independentemente do tipo de ambiente que o agente atua, o essencial é o agente ter a capacidade de interagir e de ser autónomo nesse mesmo ambiente. Para isso o agente deve possuir sensores e atuadores adequados no ambiente onde ele atuar, e competente para a execução das tarefas para as quais foi projetado.

Existem várias definições de agentes, mas uma das mais conhecidas e aceites na comunidade é a definição de Wooldridge e Jennings [Wooldridge e Jennings, 1995] que

define agente como um sistema computacional baseado em software que tem as seguintes propriedades:

- Autonomia;
- Reatividade;
- Proatividade;
- Habilidade Social.

Estas propriedades são bastantes complexas ao que pode parecer à primeira vista. A autonomia é essencial na nossa definição do agente e nunca é totalmente obtida pelo agente, pois para tal, o agente tem de ser criado e posto em funcionamento através do ser humano ou através de outro agente. A construção de um agente puramente reativo, significa que o agente é capaz de se adaptar as mudanças do ambiente sem comprometer os seus objetivos de médio ou longo prazo. O interesse é criar agentes capazes de vacilar entre o comportamento reativo e o comportamento pró-ativo, entretanto, é difícil conjugar estes dois tipos de comportamentos. [Hannenbauer et al., 2001] [Reis etal., 2001].

A proatividade é definida pela obtenção de comportamentos, através da orientação dos seus objetivos e é bastante simples de implementar em sistemas funcionais. No entanto, esta simplicidade só é verificada em ambientes estáticos, isto é, não é possível a sua alteração durante a execução de um dado procedimento ou função. Para isso, o agente tem de disponibilizar de toda a informação para poder executar esse procedimento ou função, sem duvidas no ambiente onde ele existe. Para ambientes dinâmicos e não sendo totalmente acessíveis, o agente tem de ser capaz de raciocinar e reagir a mudanças no ambiente, sendo os seus objetivos originais ainda validos. Por isso, o agente tem de ser reativo e consequentemente capaz de reagir a mudanças bruscas no ambiente.

A habilidade social de um agente pode ser definida com a capacidade de trocar mensagens de alto nível e efetuar comunicação com outros agentes ou humanos semelhantes a ele mesmo. As principais características da habilidade social de um agente é a coordenação, cooperação, competição e negociação. O agente tem de estar suficientemente apto a raciocinar acerca dos seus objetivos e do dos outros agentes presentes no ambiente, ou mesmo, ter a noção da existência deles. É necessário compreender que são agentes autónomos e que podem não partilhar os mesmos objetivos, para isso, é necessário saber negociar e cooperar com outros agentes trocando informações para conseguiram dialogar entre eles.

2.2 Atributos dos Agentes

Os agentes possuem vários atributos para exercitar as suas funcionalidades e concretizar os seus objetivos. Anteriormente foram referidas um conjunto de propriedades básicas dos agentes, sendo que essas propriedades ou atributos podem ser utilizadas para agrupar os agentes em várias classes. O próprio agente não tem que obrigatoriamente possuir todos esses atributos, mas as suas capacidades têm de estar diretamente associadas a presença delas. Sendo que a escolha dos atributos ao agente depende essencialmente da funcionalidade do agente, e as que o projetista pretende fornecer a esse mesmo agente.

2.1.1 Autonomia

A autonomia é a característica principal e mais consensual na definição do conceito de agente pelos investigadores. A autonomia refere-se ao princípio que os agentes podem agir, baseando-se nas suas próprias regras de decisão. Alguns defendem que a autonomia do agente possa aumentar consoante a proatividade desse mesmo. No entanto, o agente tem de ter a capacidade de agir por iniciativa própria, sem necessitar da intervenção do Homem e sem a necessidade de agir em virtude das mudanças do ambiente. [Wooldridge e Jennings, 1994].

A autonomia pode ser classificada em cinco tipos distintos, tais que:

- Autonomia absoluta;
- Autonomia social;
- Autonomia de interface;
- Autonomia de execução;
- Autonomia do projeto.

A autonomia absoluta define que o agente tem total controlo sobre as suas perceções, raciocínio e ações, e é pouco previsível. A autonomia social consiste em que o agente conhece os outros agentes presente no sistema e é social, sendo que ele exerce a sua autonomia em certas circunstancias. A autonomia de interface consiste quando, o agente não tem autonomia absoluta, sendo que a sua autonomia máxima possível é a autonomia respeitante a sua forma de interface com o exterior. A autonomia de execução é a liberdade que o agente possui na execução das ações no ambiente. A autonomia do projeto é o grau de autonomia do projetista do agente na sua construção, isto é, quanto maior for a autonomia do projeto, maior é a dissemelhança dos agentes.

2.1.2 Mobilidade

A mobilidade é a capacidade de um agente se poder movimentar, isto é, alterar a posição do agente para exercer funções diferentes, o que implica que o agente esteja apto para concretizar os objetivos requeridos. No entanto, este atributo pode causar sobrecarga na rede, devido a necessidade dos agentes transitaram entre diferentes posições, o que origina um aumento na dimensão do código do agente e na dimensão dos dados que esta troca com o exterior. A mobilidade dos agentes físicos refere-se à capacidade de deslocar fisicamente um agente no seu ambiente, o que por sua vez, o agente tem de estar equipado de sensores sensoriais, perceção ativa, navegação robusta, metodologias de localização própria e mapeamento do ambiente.

2.1.3 Reatividade

A reatividade tem dois significados distintos, sendo que o primeiro se refere a capacidade de o agente reagir rapidamente as mudanças no seu ambiente. O segundo refere-se à capacidade de o agente tomar as suas decisões sobre as ações a executar sem consultar nenhum modelo interno do mundo.

2.1.4 Proatividade

Este atributo representa um comportamento independente, visto que as ações tomadas pelo agente são selecionadas através dos objetivos do projetista e não as mudanças que ocorrem no ambiente do agente.

2.1.5 Comunicação

A comunicação refere-se ao dialogo entre dois ou mais agentes presentes no ambiente, ou mesmo a comunicação incluindo humanos. Os agentes necessitam de dispor de atuadores apropriados para enviar essas mensagens, de conhecimento para utilizar protocolos que permitam transportar essas mensagens e de uma linguagem que possa ser entendida por outros agentes presentes no ambiente.

2.1.6 Cooperação

Este atributo pode ser entendido como a capacidade dos agentes trabalharam em conjunto de forma a concluir tarefas de interesse comum. [Nwana, 1996]. A cooperação entre agentes é uma das principais razões para a existência de sistemas multi-agentes. O agente tem de estar habilitado a comunicar com outros agentes e possivelmente humanos e dispor de metodologias apropriadas para realizar uma boa cooperação.

2.1.7 Aprendizagem

A aprendizagem é um processo interativo em que um treinador ou projetista ensina a um ou mais agentes a executar uma tarefa individual ou cooperativa. O treinador fornece

ao agente conhecimento, através de instruções ou informações, para que o agente tenha o mínimo de capacidades ou conhecimentos para executar as tarefas e que lhe permita detetar situações de falha ou sucesso com uma maior facilidade.

2.3 Sistema Multi-Agentes

A investigação relativamente aos SMA - Sistema Multi-Agent teve inicio nos anos 80, mas destacou-se uma notoriedade digna de destaque pelos aos 90. [Wooldridge, 2002]. A longo destes anos os SMA tiveram um crescimento acentuado, criando livros, revistas e conferencia internacionais relativamente a este assunto.

Um sistema multiagentes (MAS) é um sistema computorizado composto de vários agentes inteligentes interagindo dentro de um determinado ambiente, a fim de alcançar o seu próprio objetivo individual como ilustrado na Figura 1. Para isso eles tem de estar suficientemente dotados de conhecimentos e/ou habilidades para poder alcançar as suas próprias competências e os seus próprios objetivos. Eles têm de ter a capacidade de serem autónomos para poder tomar as suas próprias decisões, para assim atuar relativamente as mudanças acontecidas no ambiente onde o próprio atua.

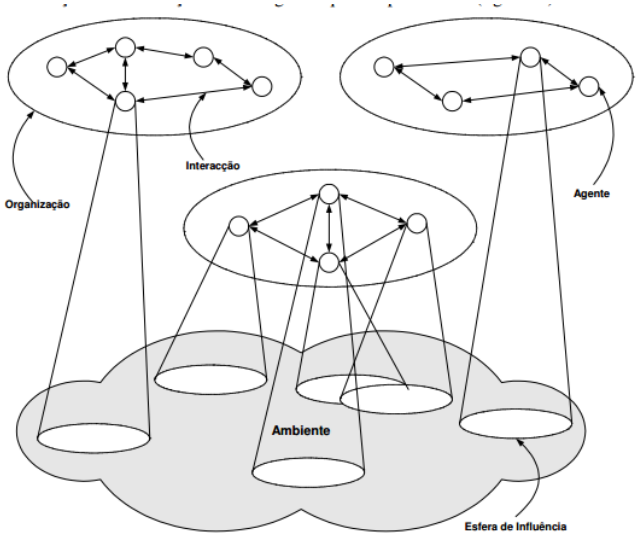

Figura 1 - Estrutura de um Sistema Multi-Agente [Girardi.R(2004)]

Um dos objetivos mais importante para a criação de sociedades de agentes, consiste em conseguir coordenar os agentes num SMA de modo a interagiram entre eles. Assim sendo, a coordenação desempenha um papel muito importante nos SMA, sendo estes sistemas inerentemente distribuídos.

Diferentes metodologias de coordenação foram propostas por vários autores dividindo-se em dois grupos: metodologias aplicáveis em domínios de agentes

competitivos *"self-interested",* isto é, agentes preocupados com o seu bem próprio, e metodologias aplicáveis a domínios contendo agente cooperativos, isto é, agentes que tem a noção de preocupação pelo bem do conjunto.

2.4 Desenvolvimento de sistemas multi-agentes

O desenvolvimento dos sistemas multi-agentes consiste em criar vários agentes com o objetivo de que comunicam entre eles para assim criar um sistema. Os agentes foram desenvolvidos no Software JADE (JAVA Agente Development Framework) através de uma linguagem de programação Java, utilizando uma linguagem de comunicação FIPA (Foundation for Intelligent Physical Agent). Após a programação dos agentes, esses mesmo foram compilados para o Raspberry Pi com o objetivo de por o sistema de transporte modular a funcionar corretamente.

2.4.1 JAVA

A linguagem de programação JAVA foi criada pela Sun Microsystems em 1995 para o desenvolvimento de programas em ambientes heterogéneos ligados em rede. O objetivo inicial era a sua utilização em sistemas isolados com quantidade mínima de memoria, isto é, em dispositivos eletrónicos para o consumidor final. Com a revolução da internet e da World Wide Web, a linguagem Java criou um interesse revolucionário por parte da comunidade, apercebendo-se do grande potencial desta linguagem na criação de paginas web com conteúdo dinâmico. [Deitel 2000] O Java é utlizado em diversas áreas, tais como, o desenvolvimento de aplicativos corporativos de larga escala, como aplicativos para dispositivos portáteis ou mesmo para a programação de agentes, sendo esta a nossa principal aplicação.

O java é composto por três elementos, tais como, a linguagem orientada a objetos, um conjunto de bibliotecas e uma máquina virtual no qual os programas são criados. O ambiente da execução foi construído com base numa maquina virtual.

O código fonte do Java é compilado para um código em bytes e em seguida pode ser interpretado por qualquer maquina virtual Java (JVM - Java Virtual Machine). Alem disso, depois de compilado para bytecode, os programas Java podem ser executados em qualquer plataforma para o qual haja um ambiente de execução Java. Desta forma, muitos mecanismos foram integrados em Java, como os Threads, comunicação entre objetos e

tratamento de eventos. Estas características em conjunto fornecem um grande poder a linguagem, principalmente para o desenvolvimento de agentes.

O Java é caracterizado por ser simples devido a linguagem de programação ser baseada em C++, removendo as características de orientação ao objeto que são raramente usados, tornando assim o Java muito mais simples. Ele é também caracterizado por ser orientado ao objeto, pois o Java suporta a escrita do software usando o paradigma orientado ao objeto, isto é, a programação orientada ao objeto é baseada na modelagem do mundo real em termos de componentes de software denominados de objetos. Um objeto consiste em dados e operações que são características que podem ser utilizados por outros objetos sem ter de reproduzir a funcionalidade daquele objeto, isto é, a reusabilidade de código. O Java é distribuído, sendo especialmente projetado para trabalhar em ambiente de redes, possuindo uma grande biblioteca de classes para a comunicação entre sistemas heterogéneos e distribuídos. É um software robusto devido a restrições impostas para evitar erros do desenvolvedor. O java tem também com características ser multithreaded, isto é, que o Java tem uma linguagem que pode ser utilizada para aplicações nas quais as múltiplas linhas de código podem ser executadas ao mesmo tempo, baseando-se em um sistema de rotinas que permite para múltiplos threads.

Relativamente ao desenvolvimento de sistemas multiagentes encontram-se varias características importantes sobre o Java, tais como:

- **Autonomia –** Um programa para ser autónomo deve ser um processo ou uma thread separada. As aplicações Java são processos separados e podem executar por um longo período de tempo. Além disso, é possível implementar um agente como uma thread separada. O importante para um agente ser autónomo é como o agente sabe que algo mudou, para isso, a forma mais natural é através de notificação por eventos, isto é, o Java disponibiliza um sistema de tratamento de eventos, utilizado nos sistemas de janelas AWT (Abstract Windowing Toolkit).
- **Mobilidade** - A mobilidade em Java é facilitada pela portabilidade do bytecode e dos arquivos JAR (varias classes agrupadas em um único arquivo para facilitar a distribuição dos programas). Em Java os códigos são enviados pela rede e executados em outra maquina. Um dos requisitos para programas com mobilidade é a habilidade de salvar o estado do processo em execução, despacha-lo e depois restaurar o processo onde for que ele seja entregue. O java possui bibliotecas pré-definidas para serem

usadas instantaneamente. Uma biblioteca é um conjunto de arquivos (.class, bytecodes).

2.4.2 JADE

O JADE – *Java Agente Development Framework* é um software totalmente implementado na linguagem Java criado em 1998 na Itália pela Telecom Itália, motivada pela necessidade de validar a recém-criada especificações FIPA. Uma das principais vantagens da utilização do JADE é que ele simplifica a implementação de SMA, através de um middleware que esta em conformidade com as especificações do FIPA e através do conjunto de ferramentas gráficas que suportam as fases de depuração a implantação. Esta estrutura provê aos programadores as seguintes características que o Jade oferece a sistemas multiagentes:

- Transporte eficiente de mensagens no formato *FIPA-ACL* dentro da mesma plataforma dos agentes;
- Compatibilidade total com as especificações FIPA;
- Conjunto de ferramentas para suporte gráfico que facilitam o trabalho de depuração e monitorização;
- Implementação de páginas brancas e amarelas, onde sistemas podem ser implementados para representar domínios e subdomínios de um grafo de diretórios.
- Bibliotecas de protocolos *FIPA* de interação que estabelecem modelos de comunicação orientados a realização de um ou mais objetos.
- Ferramentas de *debugging* que ajudam o desenvolvimento e depuração de aplicações de multiagentes.

Para alem destas características enunciadas acima, que por si mesmo já facilitam imenso o desenvolvimento de sistemas multiagentes, o Jade possui ferramentas a administração da plataforma dos agentes e o desenvolvimento de aplicações.

Na Figura 2 demonstra a arquitetura dos elementos da plataforma JADE. Esta plataforma é formada por containeres de agentes. Os agentes vivem num container, sendo estes processos do JAVA fornecidos pelo *run-time* do JADE, disponibilizando todos os serviços necessários para a sua execução. Um container especial chamado main, consiste em estabelecer o ponto de partida da plataforma, executando em primeiro todos os demais registados neles e aonde residirão os agentes de serviço da plataforma

AMS – Agent Management System, ACC e DF de maneira a que os outros containeres possam se integrar a ele.

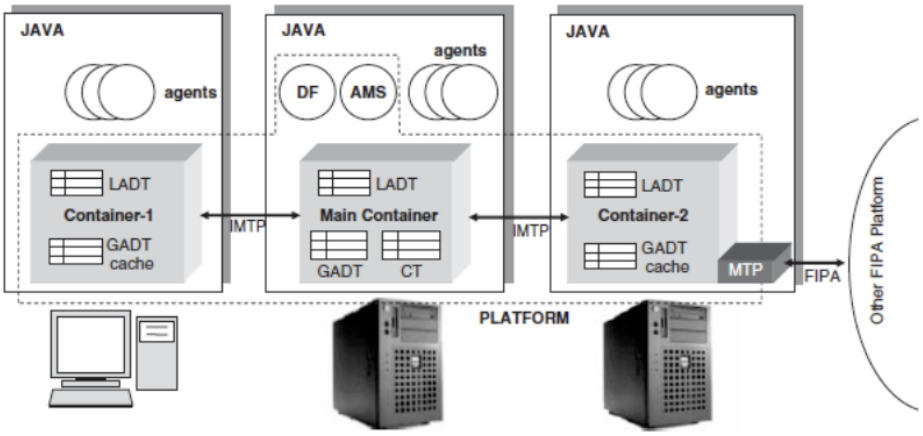

Figura 2 - Relacionamento entre os principais elementos da arquitetura. [Bellifemine, F., Caire, G., and Greenwood, D. (2007)]

Na Figura 3 constam os containeres de cada servidor, estes atuam sobre a plataforma JVM, que por sua vez garante a independência e permite que seja implementado um método de comunicação padrão entre os agentes.

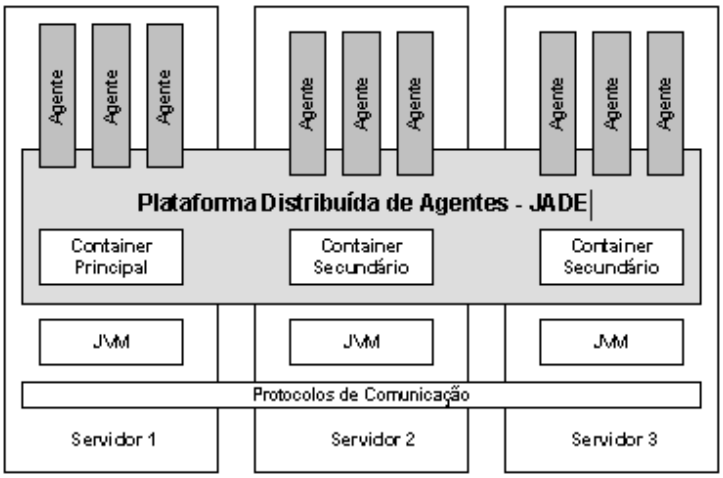

Figura 3 – Estrutura do JADE .[Bellifemine, F., Caire, G., and Greenwood, D. (2007)]

Em cada JVM é criado um container do JADE que fornece todos os serviços já mencionados para a execução e instalação de um ou mais agentes.

2.4.3 FIPA

A FIPA é uma organização sem fins lucrativos fundada em 1995 e a sua sede é residente em Genebra. Tem como objetivo criar padrões de interoperabilidade de agentes heterogéneos, interativos e de desenvolvimento de sistemas multi-agentes. Esta organização dedica-se ao desenvolvimento de uma linguagem de comunicação para agentes. [FIPA, 1999]. Sendo isto, todos os agentes têm de interagir num sistema com a

mesma linguagem, mas também atribuirão significados idênticos aos conceitos em discussão, para assim serem capazes de entender e serem entendidos pelos outros agentes.

A seguinte figura demonstra o modelo de referencia definido pela FIPA para a plataforma dos agentes.

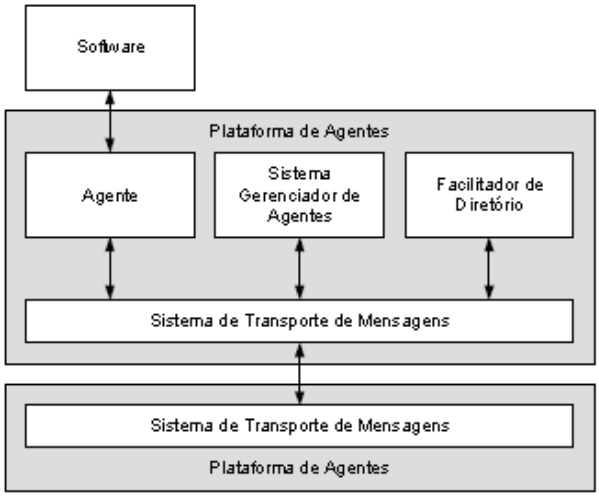

Figura 4 - Plataforma do agente. [Bellifemine, F., Caire, G., and Greenwood, D. (2007)]

O modelo é constituído por vários componentes, tais que:

- Software;
- Agente;
- Sistema de gerenciador do Agente (AMS);
- Facilitador de diretório (DF – Directory Facilitator);
- Sistema de Transporte de Mensagens (ACC – Agent Communication Channel);
- Plataforma de Agentes.

O software corresponde a todas as entidades externas que não são considerados agentes, incluindo as ferramentas corporativas, bancos de dados, componentes remotos, entre outros.

O sistema gerenciador do agente é o componente que gerência o acesso e uso da plataforma dos agentes, sendo o responsável pelo ciclo de vida do agente, pela manutenção do DF e estados dos agentes. O DF é o chamado páginas amarelas, sendo que todos os agentes podem-se registar nesse local informando o seu identificador e os serviços que presta, para assim ser identificado por outros agentes. O sistema de transporte de mensagem consiste em fornecer todos os serviços de comunicação entre os agentes dentro ou fora da plataforma.

A plataforma do agente é o ambiente onde os agentes podem ser desenvolvidos e executados, sendo composta por uma infraestrutura física e distribuída em sistemas operacionais, aplicações, componentes de gerência de agentes.

A FIPA procura garantir a interoperabilidade entre os agentes de diferentes plataformas por meio da especificação dos comportamentos externos e interface dos agentes, definindo padrões de protocolos, linguagem de conteúdo, ACL entre outros.

A FIPA-ACL é a linguagem de comunicação de agentes desenvolvida pela FIPA que se baseia também em performativas derivadas da "teoria dos atos da fala". Esta define as performativas organizadas em 4 categorias, tais como, transferência de informação, negociação, ação e gerenciamento de erros.

Na figura 5, é apresentado um exemplo de uma troca de informação entre dois agentes:

```
(inform
: sender Agente-A
: receiver Agente-B
: contente 'status=OK'
: language fipa-sl
: ontology security
)
```

Figura 5 - Troca de informação entre agentes

O Agente A envia uma mensagem informativa para o Agente B.

Capítulo 3

3 Caso de estudo

Tal como referido anteriormente, a programação baseada em agentes (entidades inteligentes) é uma tecnologia promissora para a realização do controlo de componentes Ciber-Físicos. Para isso, estes agentes serão embebidos em dispositivos de pequena dimensão e de baixo custo, mas com uma capacidade de processamento não limitada.

O sistema pretendido consiste em criar uma rede de tapetes transportadores, compostos por vários tapetes, permitindo a adição, remoção ou mesmo a troca da posição standard deles. Para isso, será criado um sistema multi-agente, em que cada transportador consiste num agente. Os transportadores deverão de ser capazes de comunicar uns com os outros através dos agentes, enviando informações relativamente à sua posição e em que estado se encontra o motor (ligado/desligado). O principal requisito no final do trabalho consiste em que o sistema seja dinâmico, reajustável e totalmente capaz de se auto-organizar. As soluções que serão desenvolvidas para cumprir os objetivos consistem em criar um sistema com a implementação de agentes de software na plataforma Raspberry Pi para realizar o controlo do sistema multi-agentes, utilizando as células Fischertechniks para assim criar CPS auto-organizados.

Com isto, será desenvolvida uma placa de interface do tipo "*shield*" como representado na figura 6 responsável por:

- fornecer energia para o motor do tapete da célula Fischertechniks e fornecer uma alimentação isolada para o Raspberry Pi;
- conectar a estrutura física do Raspberry Pi (parte de processamento) e do tapete (parte física);
- indicação através de leds do estado de operação do sistema e do estado das portas GPIO - General Purpose Input/output.

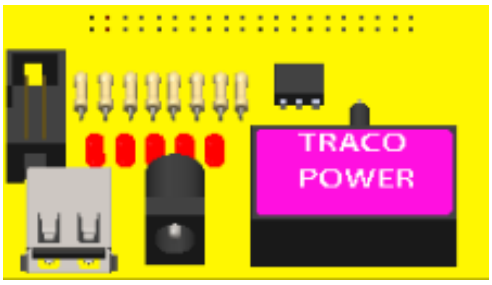

Figura 6 - Placa de interface "Shield"

3.1 Componentes Ciber-Físicos

Os CPS são uma nova geração de sistemas com capacidades computacionais e físicas que podem interagir com os seres humanos através de muitas novas modalidades.

A particularidade dos CPS é a capacidade de interagir e expandir as suas capacidades com o mundo físico através de computação, comunicação e o controlo.

Com o desenvolvimento da tecnologia e principalmente com os CPS, prevê-se um desenvolvimento desta tecnologia, como por exemplo, na condução urbana totalmente autónoma, desenvolvimento de aviões, e prótese que permitem que o cérebro receba sinais para controlar movimentos físicos, entre outras. Ao longo dos anos, os sistemas e pesquisadores da ciência da computação desenvolveram sistemas de grandes avanços na nova programação de linguagens, técnicas em tempo real de computação, métodos de visualização, sistemas de arquitetura e sistemas de software para garantir sistemas de computador com confiabilidade, segurança e tolerância a falhas. Sendo isto, o CPS é um mecanismo controlado ou monitorizado por algoritmos baseados em computador e rede, sendo totalmente integrados com a internet industrial. A tecnologia baseia-se em sistemas embarcados, computadores e software incorporado em dispositivos, cujo o objetivo é integrar a dinâmica dos processos físicos com os do software e da rede, fornecendo assim a modelagem do sistema para o seu funcionamento. Na figura 7 está representada a pirâmide da arquitetura dos CPS.

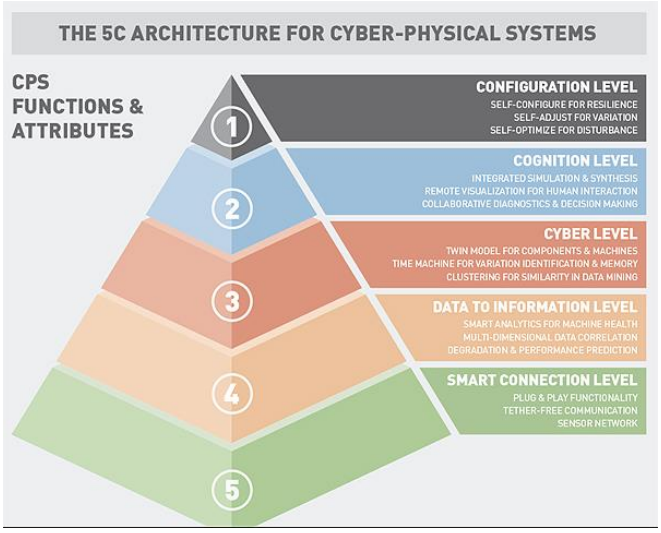

Figura 7 - Arquitetura dos CPS. [Behrad, 2015]

3.2 Célula Fischertechniks de tapetes

Na figura 8 esta ilustrada uma célula Fischertechnik, dotada de uma correia transportadora que opera através de um motor de 24V DC. A deteção da peça no inicio e no fim do tapete é feita através de um sensor de barreira foto-elétrico. De forma semelhante ao motor de corrente continua, os sensores operam a tensão nominal de 24V, o que torna estas células bastantes flexíveis, permitindo assim serem compatíveis diretamente com um Controlador Lógico Programável (PLC).

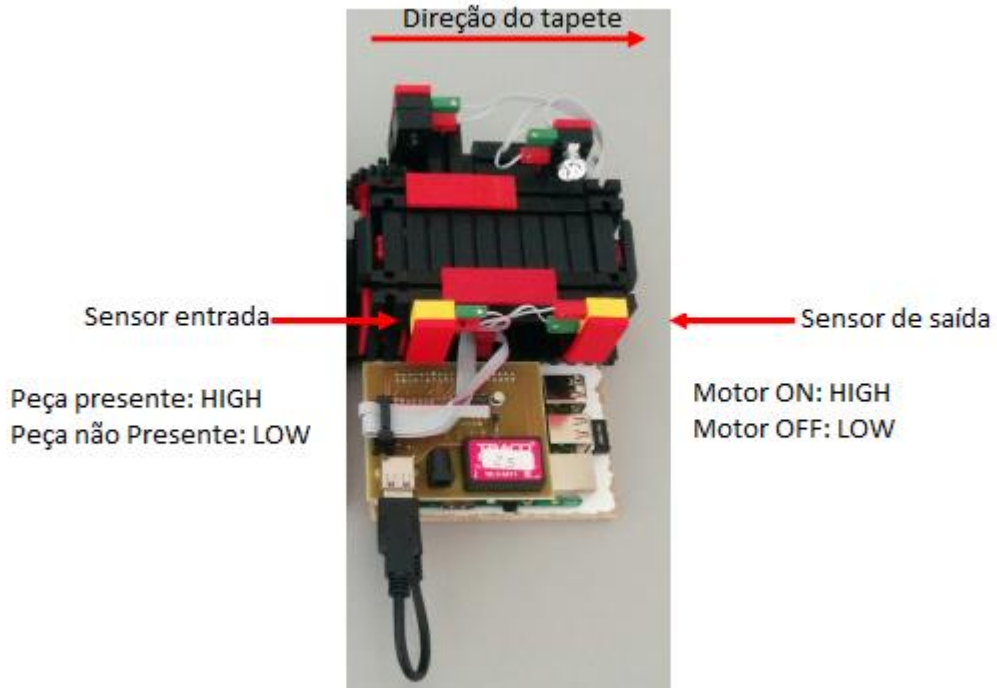

Figura 8 - Componente Ciber-Físicos com célula Fischertechniks

3.3 Raspberry

O Raspberry Pi é um minicomputador de baixo custo e de tamanho de um cartão de crédito, ele permite a conexão de um monitor de computador ou mesmo de uma televisão, permite o uso de um teclado e de um rato. O Raspberry foi desenvolvido no Reino unido pela fundação Raspberry Pi iniciando a sua venda em fevereiro de 2012. O modelo Raspberry Pi modelo 3 (3ª geração) foi o último a ser lançado em fevereiro de 2016, incluindo um processador quad-core ARMv8 de 1.2Ghz 64bit, com 1GB RAM. O hardware está integrado numa única placa. Ele é compatível com sistemas operativos baseados em Linux, com o sistema operativo armazenado num cartão SD. O Raspberry inclui várias linguagens de programação, tais como, Python, Scratch, Java, etc. Ele permite fazer tudo o que se desejaria fazer num computador desktop, desde navegar na internet, reprodução de vídeo e som, processamento de texto, etc. Além destas inúmeras funcionalidades do Raspberry, ele tem a capacidade de interagir com o mundo exterior e

tem sido usado numa ampla gama de projetos, como por exemplo, a criação de sistemas multiagentes.

Na figura 9 está ilustrada a entrada HDMI, alimentação do Raspberry, o slot do cartão de memória, as portas GPIOS, a porta RCA, áudio, entrada USB, e entrada Ethernet.

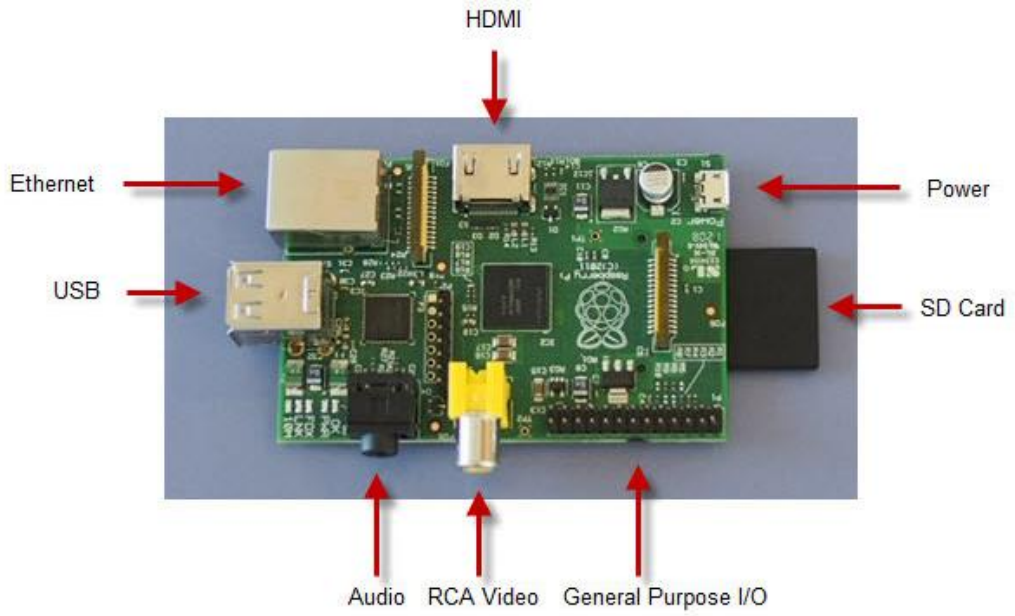

Figura 9 – Componente ciber do sistema RaspberryPi

O objetivo no final do trabalho é desenvolver os agentes na plataforma JADE e colocar os agentes desenvolvidos em cada Raspberry Pi para assim criar CPC - Cíber-Physical Components. Posteriormente, tendo os agentes implementados no CPC podemos agregá-los e criar um CPS, constituindo assim um sistema de transporte modular e reconfigurável.

Capítulo 4

4 Engenharia da solução

4.1 Desenvolvimento do agente tapete

Neste trabalho, para realizar o controlo lógico dos módulos dos tapetes, foi adotada a tecnologia dos agentes implementados no Raspberry Pi. A placa do Raspberry Pi é alimentada através de um cabo USB padrão, com uma fonte de alimentação de 5V e as portas General Purpose Input Output (GPIO) são compatíveis com uma alimentação a 3.3V.

O sistema de transporte é dividido em duas partes simbióticas, ou seja, a parte lógica e a parte física. A parte lógica é a capacidade de processamento proveniente dos agentes em execução no Raspberry Pi, constituindo a parte Cyber do sistema. Enquanto que a parte física é toda a parte constituída pelas células Fischertechniks.

Na figura 10 é representado o sistema Cyber-Físico, distinguido as duas partes do sistema.

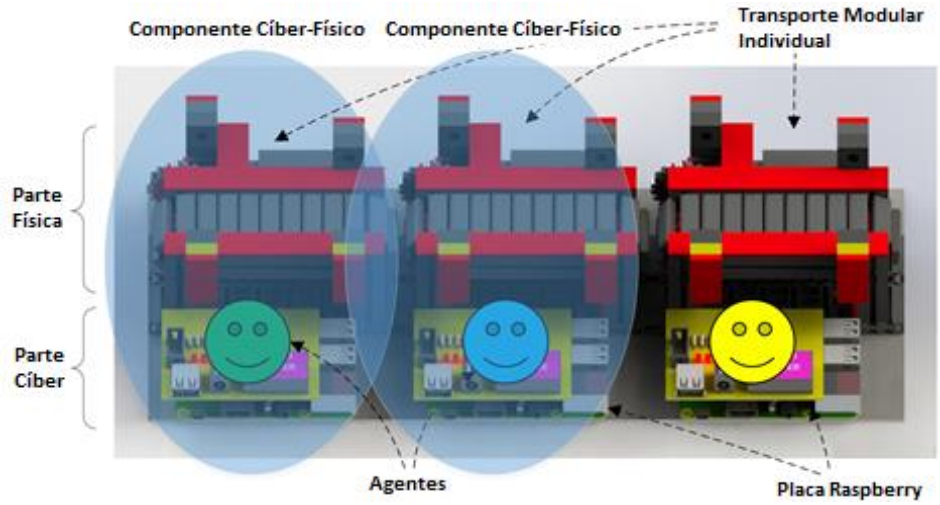

Figura 10 – Componente ciber-fisico indivual

Para a criação dos agentes no nosso sistema, utilizou-se a plataforma JADE para a implementação dos agentes que tem como objetivo controlaram os tapetes. Uma das características mais importantes da utilização desta tecnologia é a possibilidade de programar uma única vez e implementar em vários componentes Ciber-Físicos com a mesma programação. Mesmo assim, esta característica só se torna válida e plenamente explorada quando a criação, o desenvolvimento e a programação do agente esteja devidamente projetado para ser tão genérico o quanto possível.

Assim sendo, o comportamento de cada agente que individualmente controla um tapete, tem um processo de iniciação simples que envolve registar o agente com as propriedades dele no DF. Esta fase de iniciação é finalizada com o lançamento de um comportamento que é responsável por lidar com a troca de mensagens e do processo de auto-organização. O comportamento de cada agente baseia-se essencialmente no controlo do motor do tapete, consoante o estado (High ou Low) do sensor de entrada e a saída dos respetivos tapetes e das mensagens recebidas dos agentes antecedentes e posterior ao atual. Na seguinte figura 9 está ilustrada a sequência do procedimento das intervenções que deve tomar cada agente consequentemente às informações recebidas. Na verdade, cada agente deve informar o agente adjacente de duas formas, quando o tapete tem uma peça no sensor de entrada (P4) deve informar o tapete anterior (t4), caso exista, para parar o motor. Da mesma forma, quando o tapete tiver uma peça no sensor de saída (p7), este tem de enviar uma mensagem para o tapete seguinte (t7), caso haja, para ligar o seu motor de modo a receber a peça. Na seguinte figura 11 está representado o controlo lógico individual do agente.

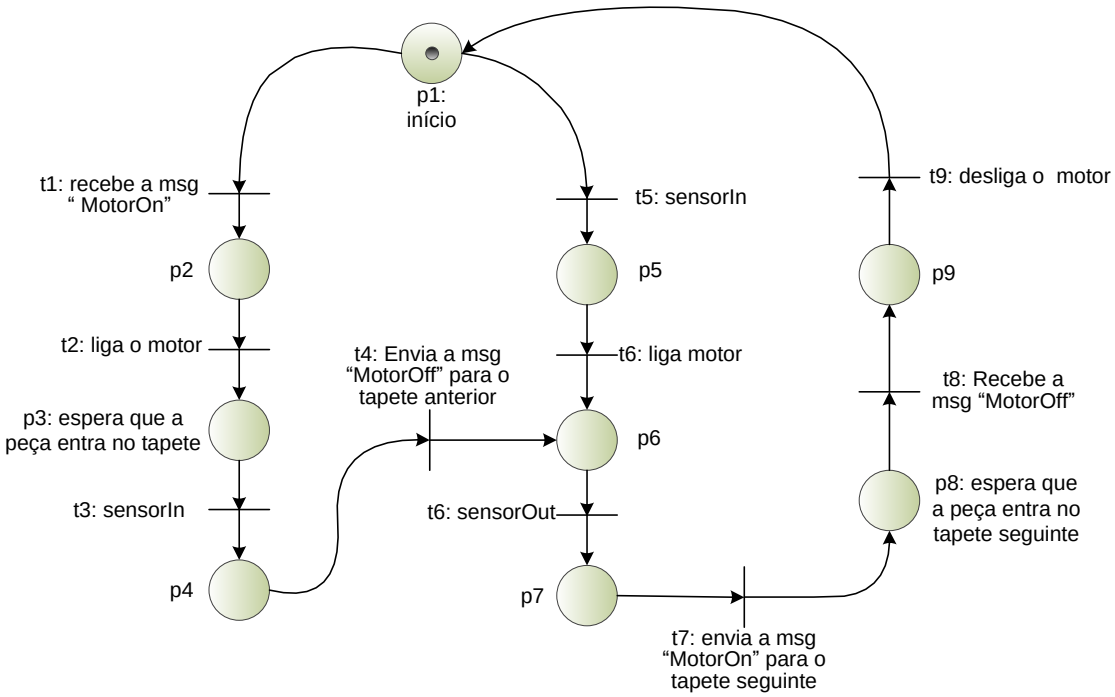

Figura 11 - Controlo lógico para o agente individual do tapete

A seguir na Figura 12 está representado o diagrama de protocolo do Agent Communication Language (ACL).

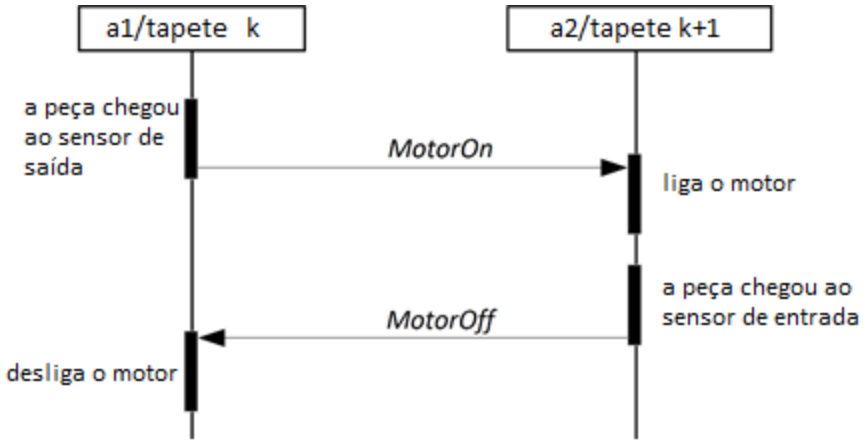

Figura 12 - Protocolo de interação para sincronizar os diferentes tapetes individuais

A mensagem *MotorOff* (t8) é utilizada para informar o agente anterior que a peça já chegou ao sensor de entrada no tapete posterior e que o tapete anterior está em conformidade para desligar o seu tapete (t9). Enquanto que a mensagem *MotorOn (t7),* é utilizada para informar o agente posterior para ligar o seu motor porque o agente anterior tem uma peça no sensor de saída. Estas mensagem contêm também informações adicionais, tais como, indicação da posição do agente que envia a mensagem, com o objetivo de cada agente saber se tem de realizar alguma tarefa ou se não lhe diz respeito. Os agentes além destas duas mensagens, trocam outros tipos de mensagens entres eles durante os seus processos de cooperação, nomeadamente “swapFound”, “IAmASwap”, “ImHere”, “position”, “uptadeposition”, “positionOut”, “fim”, “swapFoundConveyor”. Estas mensagens, que constituem várias fases e situações dos comportamentos dos agentes será profundamente explicado durante a descrição dos mecanismos de auto-organização e o código desenvolvido encontra-se nos anexos do relatório de projeto.

4.2 Arranque inicial do sistema

No arranque inicial do sistema, nenhum dos agentes está consciente da posição em que se encontra. Quando a primeira peça é colocada num dos tapetes e o agente correspondente não sabe a sua posição, o agente assume o seu token valor 0. Em seguida, quando a peça chega ao sensor de saída desse mesmo agente, é enviada uma mensagem a todos os tapetes do sistema. Nesta fase, os tapetes que tiveram o token valor 0 irão todos ligar os motores dos tapetes. Note-se que os tapetes que já conheceram as suas posições e não foram o tapete seguinte onde se encontra a peça irão ignorar a mensagem, sendo que não irão ligar os motores. O agente que recebe a peça por sua vez, e que não sabe a

sua posição, irá atualizar a sua posição em conformidade a posição recebida do anterior. Na figura 13 está representado o controlo lógico do sistema no arranque inicial.

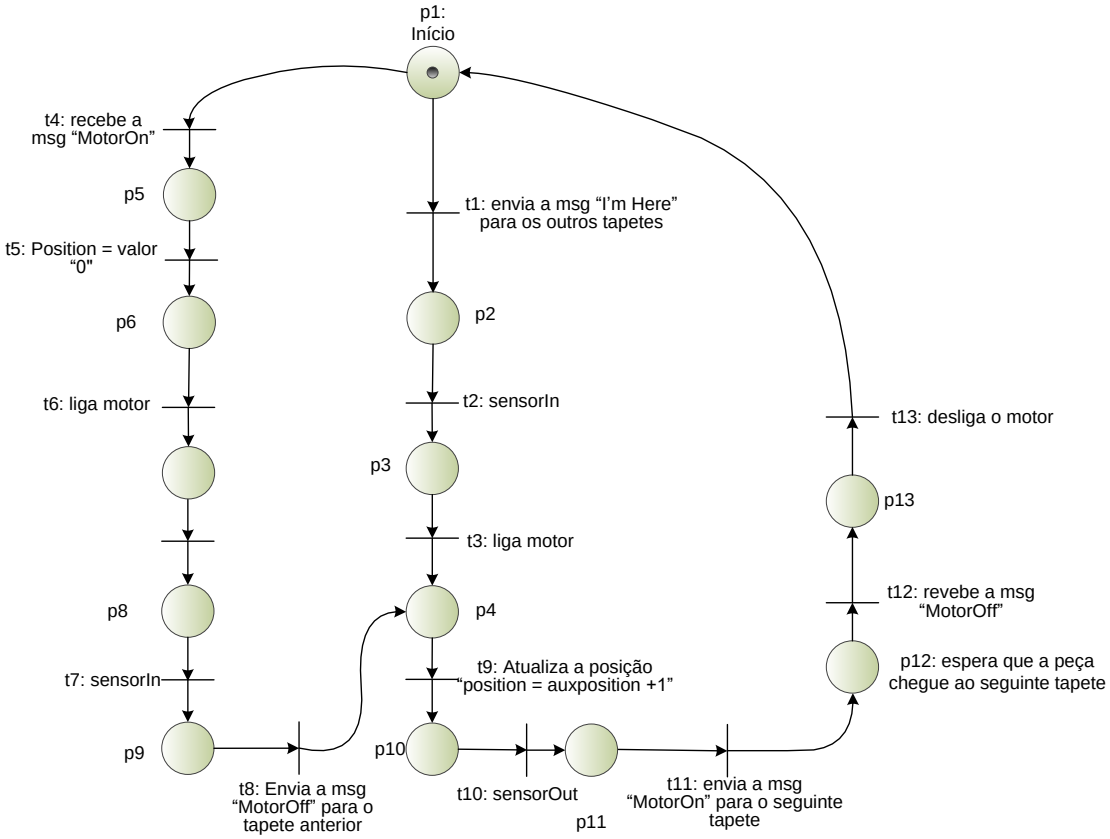

Figura 13 - Controlo lógico inicial do sistema do tapete

4.3 Desenvolvimento dos Agentes no Raspberry Pi

Após a implementação dos agentes, estes mesmos devem ser implementados e executados em cada placa do Raspberry Pi. Devidas as características inerentes do Jade, o main-container deve ser iniciado antes da execução dos agentes. Apesar do facto de uma das placas do Raspberry Pi poder ser selecionada a acolher este container principal, um sistema host baseado em nuvem é selecionado. Esta decisão também permite atingir mais funções, tais como, remover qualquer transportador, libertando o utilizador a partir da constante necessidade de assegurar o main-container. É também digno de mencionar que nenhum centro de mecanismo é implementado na nuvem, ou seja, o sistema é implantado na nuvem usando uma auto-organização descentralizada, sem fornecer qualquer topologia do sistema, tamanho ou mesmo a posição de transporte para os agentes. A implementação real de agentes no Raspberry Pi é simples, sendo apenas necessário descarregar o pacote (ou seja, o ficheiro .jar contendo a instanciação do agente) em um diretório do sistema. Apesar disso, vários preparatórios de trabalho na placa do

Raspberry Pi são aconselháveis, ou seja, a instalação de um Java Runtime Environment (JRE), a instalação de uma API de Java para permitir que o agente acesa às portas GPIOs e/ou a informação do sistema. A implementação atual usa API PI4J Java (http://pi4j.com/) que liga o kernel do Raspberry Pi em compatibilidade com os métodos do Java. Além de permitir o controlo de GPIOs, esta API também permite o acesso a outras funções, tais como a comunicação, reunindo as informações do sistema/rede e a criação de listeners (ouvintes). Para ter acesso ao HW do Raspberry Pi, o agente precisa ter direitos de *root*. O acesso do agente ao HW é configurado durante o seu comportamento de inicialização, onde é requerido as portas GPIOs usando o seguinte enxerto de código:

```
myMotor = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04, Pinstate.LOW);
inSensor = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_06, "inSensor");
outSensor = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05, "inSensor");
```

Os listeners também são adicionados para orientar ações do agente desencadeada pela mudança do estado na entrada e na saída dos sensores de luz fotos-elétricos. A principal função dos listeners é estar permanentemente atentos a qualquer mudança de estado dos sensores de entrada ou saída, ou mesmo, do estado do motor, ligado ou desligado. O seguinte enxerto de código exemplifica o listener para o sensor de entrada.

```
inSensor.addListener(new GpioPinListenerDigital() {
        @Override
        public void handleGpioPinDigitalStateChangeEvent(
                GpioPinDigitalStateChangeEvent event) {
        // include here the code to be executed by Raspberry Pi
        // when the sensorIn signal changes its state
        }
});
```

Capítulo 5

5 Desenvolvimento dos mecanismos de auto-organização

Um problema importante durante a operação dos módulos dos tapetes, CPS, é garantir a implementação dos mecanismos que suporta a compatibilidade e a auto-organização do sistema de transporte, isto é, sem a necessidade de parar a linha, reprogramar e/ou reiniciar os componentes individuais.

Para este fim, esta seção consiste em detalhar a implementação dos mecanismos de auto-organização para apoiar os sistemas a operação em ambientes evoluíeis, ou seja, a adição de um novo tapete, removendo um tapete avariado ou mesmo a troca deles. O mecanismo global de auto-organização é, em seguida, construído por a composição de regras simples (semelhantes ao que acontece na natureza).

5.1 Plug-in e plug-out de tapetes

O plug-in de um novo transportador é realizado por o agente durante a sua fase de inicialização, registando suas habilidades e envio de uma mensagem "*I'm Here*" para os outros agentes, informando que ele está pronto para executar as suas funções. Os outros agentes, recebendo esta mensagem, são capazes de atualizar a quantidade de CPC (Cyber-physical-Component) existentes no sistema. A introdução de um novo tapete pode acontecer em qualquer uma posição do sistema, ou seja, no início, no fim ou mesmo no meio, sendo necessário que a programação do mecanismo seja simples e automática para a deteção de um novo tapete na linha de transporte, como ilustrado na figura 14. A situação mais simples para o sistema é quando o tapete é introduzido no final da linha, sendo apenas necessário a atualização do número total de tapetes. De facto, durante a fase de operação normal, os agentes mantêm a sua posição até eles não detetaram qualquer alteração da ordem.

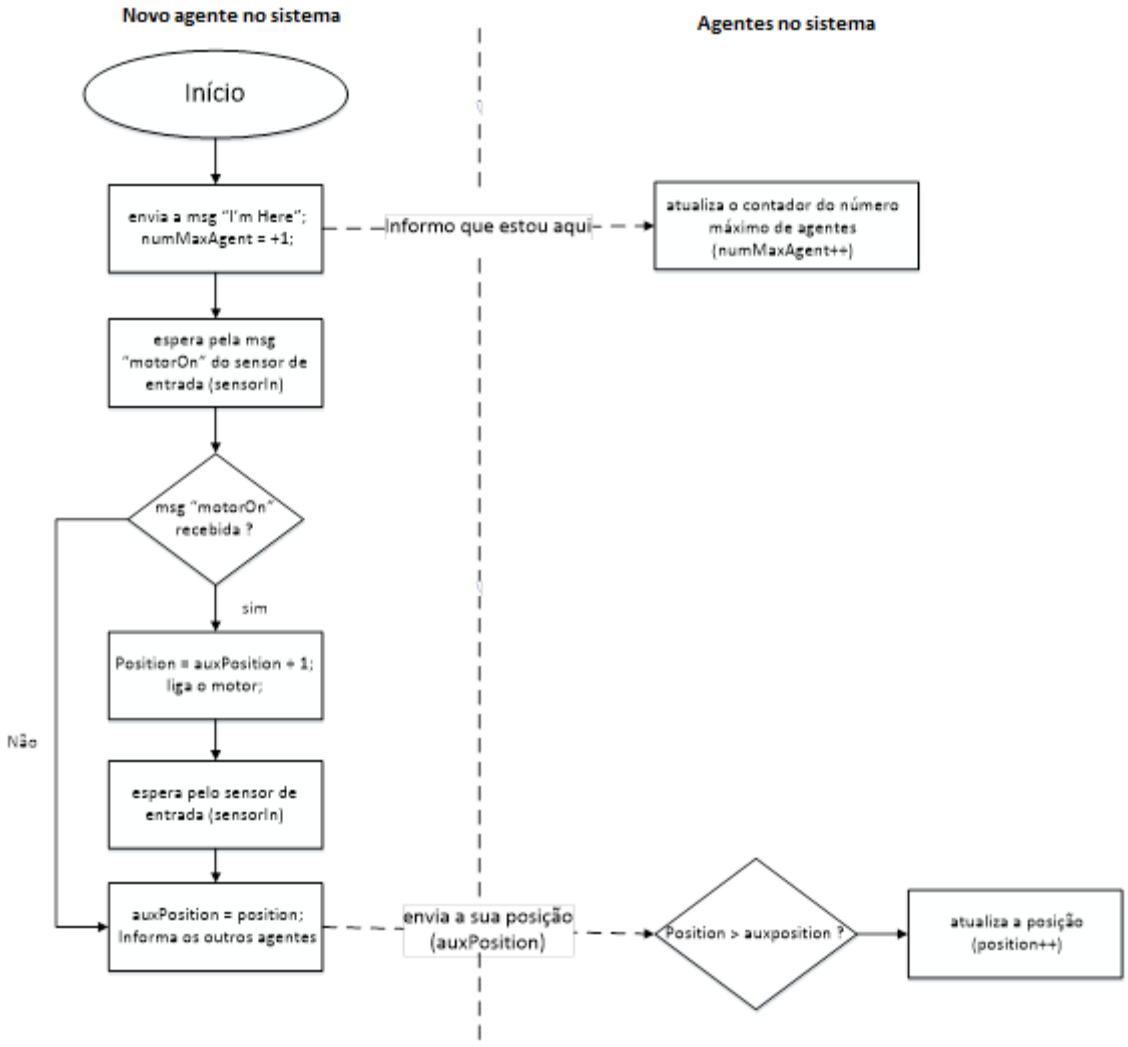

Figura 14 - Mecanismo para o plug-in de um novo tapete no sistema automatizado.

Quando o tapete é introduzido no início do sistema e uma peça é colocada para ser transportada, o tapete inicia como referido anteriormente, com o token a 0, sendo que, ainda não é conhecida a posição dele, irá por sua vez atualizar a sua posição para 1 e consequentemente enviará uma mensagem "updatePosition" para os outros agentes que tenham recebido uma mensagem "ImHere", sabendo estes mesmos que há um novo tapete e a sua posição é igual ou superior ao agente que enviou a mensagem "updatePosition" com o conteúdo da sua posição, estes mesmo atualizam a sua posição para mais uma. Da mesma forma, a remoção de um tapete é transmitida pelo agente designado antes da remoção do sistema enviando uma mensagem de "positionOut" juntamente com a posição dele, sendo que os tapete que tiverem uma posição maior do que a dele, atualizam consequentemente a posição para menos uma, como ilustrado na Figura 15.

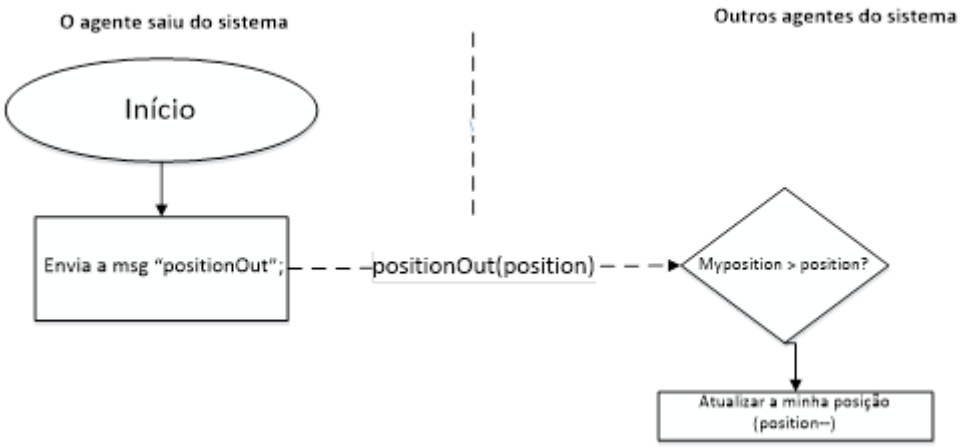

Figura 15 - Mecanismo para o plug-out de um tapete no sistema

5.2 Alteração da ordem da sequência dos tapetes

O mecanismo de "*swap*" requer uma abordagem ligeiramente diferente, uma vez que a quantidade dos tapetes é mantida e devem ser adquiridas as novas posições. Desta forma, o mecanismo de auto-organização compreende várias etapas para apoiar a troca dos tapetes, como ilustrado na figura 16.

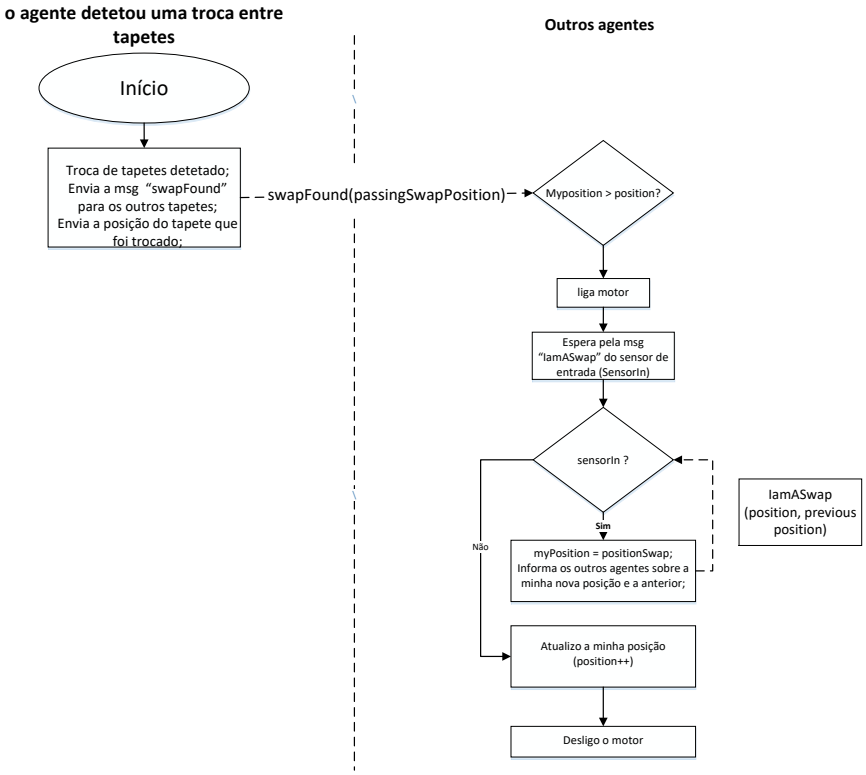

Figura 16 - Mecanismo para o swap automático dos tapetes

A primeira etapa está relacionada com a fase de deteção. Em condições normais de funcionamento, quando uma peça se encontra no sensor do fim do tapete, uma mensagem "MotorOn" é transmitida para todos os agentes existentes no sistema, mas é

unicamente processada pelo o agente transportador posterior à localização da peça. Quando o agente com a peça no final do tapete detetar que a peça não atingiu o tapete seguinte depois de 5 segundos (por meio de um timeout), transmite uma mensagem "swapFound" que alerta para uma troca da ordem standard dos transportadores. Desta forma, todos os transportadores que têm uma posição maior à posição atual da peça, por sua vez, irão ligar os motores. Quando um transportador recebe a peça no seu sensor de entrada, ele irá atualizar a sua posição atual no sistema e transmitir uma mensagem "IamASwap" com a sua posição anterior e a sua nova posição (atualizada através da mensagem recebida anteriormente "swapFound"). Este processo é repetido cada vez que um transportador troque a sua posição standard do sistema, para assim, os transportadores se auto-organizarem para o sistema ficar funcional.

O uso de agentes e princípios de auto-organização permite, de forma distribuída e descentralizada, governar este sistema de transporte utilizando um conjunto simples de regras, de modo ao sistema se manter funcional. Além disso, a utilização de agentes permite chegar verdadeiramente a um conceito de produção interligada. O objetivo passou sempre por desenvolver os vários sistemas de auto-organização por partes, sendo o primeiro objetivo fazer com que a peça percorresse o primeiro tapete até ao ultimo tapete. Depois foi desenvolvido o sistema de auto-organização, eliminando um tapete do sistema, que por sua vez, os agentes tinham de atualizar o número máximo de tapetes existente no sistema e atualizar as posições dos tapetes posteriores ao retirado para menos 1 posição. E por fim e não menos importante, foi desenvolvido o sistema de "swap", trocando 2 ou mais tapetes de modo a que os tapetes se auto-organizassem automaticamente, atualizando as suas novas posições.

Capítulo 6

6 Conclusão

Este relatório de projeto descreve o desenvolvimento de um sistema de transporte modular auto-organizado de componentes ciber-físicos. Este sistema foi desenvolvido através da construção de componentes ciber-físicos, sendo que a parte física é constituída por uma correia transportadora, operando com um motor DC 24V e com sensores de entrada e saída fotoelétricos. Enquanto que a parte lógica foi desenvolvida no Raspberry Pi que consistiu num minicomputador onde os agentes foram implementados.

Acoplaram-se vários componentes ciber-físicos, permitindo transportar peças do ponto A ao ponto B. Posteriormente foi desenvolvido um sistema transportador de peças com mecanismo de auto-organização através da criação de um sistema multi-agente, com paradigmas distribuídos e inteligentes.

A programação dos agentes foi desenvolvida em linguagem Java usando a plataforma JADE para desenvolver os agentes. O desenvolvimento dos agentes consistiu em criar um mecanismo de auto-organização através de regras simples, que permite o funcionamento do sistema dinamicamente, incluindo, como propriedades a possibilidade de adicionar e/ou remover transportadores, ou mesma a alteração da sequência destes mesmo, mantendo a operacionalidade e dinamismo do sistema automaticamente, sem a necessidade de parar, reprogramar ou mesmo reiniciar o sistema.

Como trabalho futuro poderá ser melhorado o mecanismo de auto-organização e desenvolver uma aplicação em Android para visualizar o estado atual do sistema.

Bibliografia

[ACATECH (2011)] Cyber-Physical Systems: Driving force for innovation in mobility, health, energy and production. Technical report, ACATECH German National Academy of Science and Engineering.

[Barbosa, J., Leitão, P., Adam, E., and Trentesaux, D. (2002)] Dynamic self-organization in holonic multiagente manufacturing systems: The ADACOR evolution. Computers in Industry, 66, 99-111.

[Bauer, H., Baur, C., Camplone, G., and et. al. (2015)] Industry 4.0: How to navigate digitization of the Manufacturing sector. Technical report, McKinsey Digital.

[Behrar,Bagheri, (2015)] NS for Intelligent Maintenance Systems, University of Cincinnati.
[Bellifemine, F., Caire, G., and Greenwood, D. (2007)] Developing Multi-Agent Systems with JADE. Wiley.

[Bussmann, S. and Schild, K. (2000)] Self-organizing Manufacturing control: An industrial application of agent technology. In Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'00), 87{94.

[Edward A. Lee. 2006] Cyber physical Systems – Are Computing Foundation Adequate. Departement of EECS, UC Berkeley.

[EFFRA (2013)] Factories of the Future, Multi-Annual Roadmap for the contractual PPP under H2020. Technical report, European Commission.

[ElMaraghy, H. (2006)] Flexible and reconfigurable manufacturing systems paradigms. International Journal of Flexible Manufacturing Systems, 17, 261-271.

[Ferber, J. (1999)] Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison-Wesley.

[FIPA, 2002] FIPA Abstract Arquitecture Specification. Standard of the Foundation for Intelligent Physical Agents.

[Girardi.R(2004), Engenharia de Sotware baseada em Agentes, PUC Rio]

[IEC - International Electrotechnical Commission (2012a)] IEC 61131: Programmable controllers Part 3: Programming languages.

[IEC - International Electrotechnical Commission (2012b)] IEC 61499: Function blocks Part 1-4.

[Jacobi, S.Hahn, C.Raber, D. 2010] Integration of multiagente systems and service oriented architectures in the steel industry.

[Kagermann, H., Wahlster, W., and Helbig, J. (2013)] Securing the future of German manufacturing industry: Recommendations for implementing the strategic

initiative INDUSTRIE 4.0. Technical report, ACATECH German National Academy of Science and Engineering.

[Lee, E.A. (2008)] Cyber physical systems: Design challenges. In Proc. of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08), 363369.

[Leitão, P. (2009a)] Holonic rationale and bio-inspirationon design of complex emergent and evolvable systems. In A. Hameurlain, J. Kung, and R. Wagner (eds.), Transactions on Large Scale Data and Knowledge CenteredSystems, Lecture Notes in Computer Science, volume 5740, 243{266. Springer.

[Leitão, P. (2009b)] Agent-based distributed Manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence, 22(7), 979{991.

[Leitão, P., Karnouskos, S., and Colombo, A.W. (2016)] Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. Computers in Industry, in press.

[Leitão, P. and Restivo, F. (2009)] ADACOR: A holonic architecture for agile and adaptive manufacturing control. Computers in Industry, 57(2), 121, 130.

[Miller, P. (2007)] The Genius of Swarms. NationalGeographic.

Anexo

Em seguida transcreve-se o código implementado no sistema

```
import java.text.SimpleDateFormat;
import java.util.Date;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.DataStore;
import jade.core.behaviours.SimpleBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;

public class Raspberry extends Agent {

Codec codec = new SLCodec();
DataStore tokenStore = new DataStore();

	protected void setup() {
	SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

	DFAgentDescription dfd1 = new DFAgentDescription();
	ServiceDescription sd1 = new ServiceDescription();
	sd1.setType("Raspberry");
	sd1.setName("");
	dfd1.addServices(sd1);
		try {
			DFService.register(this, dfd1);
			System.out.print(sdf.format(new Date()));
			System.out.println(dfd1);
			} catch (Exception e)
			{	System.out.print(sdf.format(new Date()));
				System.err.println(" - Raspberry" + getLocalName() + ":
				registration with DF unsucceeded.");
		}
	SimpleBehaviour raspberryBehaviour = new RaspberryBehaviour(this);
	addBehaviour(raspberryBehaviour);
	raspberryBehaviour.setDataStore(tokenStore);

	DFAgentDescription[] result = null;
	DFAgentDescription dfd2 = new DFAgentDescription();
```

```
		ServiceDescription sd2 = new ServiceDescription();
		dfd2 = new DFAgentDescription();
			sd2.setType("Raspberry");
		// sd2.setName(id);
		dfd2.addServices(sd2);
			try {
				result = DFService.search(this, dfd2);
				} catch (FIPAException fe) {
				fe.printStackTrace();
				}
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);

		for (int i = 0; i < result.length; i++) {
			changeMessage.addReceiver(result[i].getName());
			}
			changeMessage.setProtocol("I'm here");
			changeMessage.setContent("1");
			send(changeMessage);
			}
		@Override
		protected void takeDown() {
			super.takeDown();

			DFAgentDescription[] result = null;
			DFAgentDescription dfd1;
			ServiceDescription sd1;

			dfd1 = new DFAgentDescription();
			sd1 = new ServiceDescription();
			sd1.setType("Raspberry");
			dfd1.addServices(sd1);

			try {
				result = DFService.search(this, dfd1);
			} catch (FIPAException fe) {
				fe.printStackTrace();
			}
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);

		for (int i = 0; i < result.length; i++) {
			changeMessage.addReceiver(result[i].getName());
		}

changeMessage.setProtocol("positionOut");
changeMessage.setContent(tokenStore.get("myposition").toString());
```

```
try {
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        DFService.deregister(this, dfd);
        System.out.println("[Deregistering successful]");

} catch (Exception e) {
        System.out.println("[Problems deregistering agent]");
}
send(changeMessage);
System.out.println("[Agent Killed]");
}
}
```

Behaviour

```
import java.security.acl.LastOwnerException;
import java.sql.Time;
import java.text.SimpleDateFormat;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalInput;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;
import com.pi4j.io.gpio.event.GpioPinDigitalStateChangeEvent;
import com.pi4j.io.gpio.event.GpioPinListenerDigital;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.tools.logging.ontology.GetAllLoggers;
import java.util.Calendar;
import java.util.Date;
import java.util.Vector;

public class RaspberryBehaviour extends SimpleBehaviour {

        final GpioController gpio = GpioFactory.getInstance();
        final GpioPinDigitalOutput myLed1, myLed2, myLed3, myLed4, myMotor;
        static GpioPinDigitalInput inSensor, outSensor;
```

```
MessageTemplate mt =
MessageTemplate.or(MessageTemplate.MatchProtocol("swapFound"),
MessageTemplate.or(
MessageTemplate.MatchProtocol("position"),
MessageTemplate.or(MessageTemplate.MatchProtocol("fim"),
MessageTemplate.or(MessageTemplate.MatchProtocol("positionOut"),
MessageTemplate.or(MessageTemplate.MatchProtocol("MotorOn"),
MessageTemplate.or(MessageTemplate.MatchProtocol("MotorOFF"),
MessageTemplate.or(MessageTemplate.MatchProtocol("verificarPosition"),
MessageTemplate.or(MessageTemplate.MatchProtocol("updateposition"),
MessageTemplate.or(MessageTemplate.MatchProtocol("swapFoundconveyor1",
MessageTemplate.or(MessageTemplate.MatchProtocol("IAmASwap"),Message
Template.MatchProtocol("I'm here")))))))))));

Boolean done = false;
Codec codec = new SLCodec();
DFAgentDescription[] result = null;
ACLMessage msgIn = myAgent.receive(mt);
int count = 0;
int position = 0;
int myposition = 0;
int auxposition = 0;
int NumMaxAgent = 0;
int NumMaxAgent2 = 0;
int positionfound = 0;
AID conveyorAnterior;
long waitingTime = 0;
int swapfound = 0;
int swapfound1 = 0;
int passingSwapPosition = 0;
int passingSwapPosition1 = 0;
int positionSwap = 0;
long sentTime, receiveTime;
int motorOn = 0;
int foundSwap = 0;
int positionSend = 0;
int runconveyor = 0;
int motorOff = 0;

public RaspberryBehaviour(Agent Raspberry) {
        super(Raspberry);

        myMotor = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04, "My
        Motor", PinState.LOW);
        myLed1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_00, "My Led
        on GPIO0", PinState.LOW);
        myLed2 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_02, "My Led
        on GPIO2", PinState.LOW);
        myLed3 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "My Led
        on GPIO3", PinState.LOW);
```

```
                myLed4 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_07, "My Led
                on GPIO7", PinState.LOW);
                inSensor = gpio.provisionDigitalInputPin(RaspiPin.GPIO_06,
                "inSensor");
                outSensor = gpio.provisionDigitalInputPin(RaspiPin.GPIO_05,
                "outSensor");

                SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

        if(outSensor.getState().equals(PinState.LOW)&&(inSensor.getState().equals(Pin
        State.LOW) && (myMotor.getState().equals(PinState.LOW))))
        {System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + ": Nenhuma
        peça no tapete!");
        }

inSensor.addListener(new GpioPinListenerDigital() {
        publicvoidhandleGpioPinDigitalStateChangeEvent(GpioPinDigitalStateChange
        Event event) {
                if(event.getState().equals(PinState.HIGH)&&(position>1)&&myMotor.g
                etState().equals(PinState.LOW)) {
                swapfound1 = 1;
                        if (swapfound1 == 1) {
                                System.out.print(sdf.format(new Date()));
                                System.out.println(" - swapfound with conveyor 1.");
                                DFAgentDescription dfd2 = new DFAgentDescription();
                                ServiceDescription sd2 = new ServiceDescription();
                                dfd2 = new DFAgentDescription();
                                sd2 = new ServiceDescription();
                                sd2.setType("Raspberry");
                                dfd2.addServices(sd2);
                                try {
                                        result = DFService.search(myAgent, dfd2);
                                        } catch (FIPAException fe) {
                                                fe.printStackTrace();
                                }
ACLMessage changeMessage = newACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);

        for (int i = 0; i < result.length; i++){
                changeMessage.addReceiver(result[i].getName());
        }
        passingSwapPosition1 = position;
        String swapPosition1 = Integer.toString(passingSwapPosition1);
        System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - Swap
position"+ passingSwapPosition1 + "with conveyor 1.");
changeMessage.setProtocol("swapFoundconveyor1");
changeMessage.setContent(swapPosition1);
```

```
myAgent.send(changeMessage);
swapfound1 = 0;
}
if (event.getState().equals(PinState.HIGH) && (position == 0)) {
	waitingTime = 0;
	String position1 = Integer.toString(position);
	DFAgentDescription[] result = null;
	DFAgentDescription dfd2 = new DFAgentDescription();
	ServiceDescription sd2 = new ServiceDescription();
	dfd2 = new DFAgentDescription();
	sd2 = new ServiceDescription();
	sd2.setType("Raspberry");
	dfd2.addServices(sd2);
	try {
		result = DFService.search(myAgent, dfd2);
		} catch (FIPAException fe) {
		fe.printStackTrace();
		}
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage=newACLMessage(ACLMessage.INFORM);
	for (int i = 0; i < result.length; i++) {
		changeMessage.addReceiver(result[i].getName());
	}
changeMessage.setProtocol("position");
changeMessage.setContent(position1);
myAgent.send(changeMessage);
}
if (event.getState().equals(PinState.HIGH) && (position == 1)) {
	myMotor.high();
	// System.out.print(sdf.format(new Date()));
	// System.out.println(
	// " - Raspberry" + myAgent.getLocalName() + ": Peça no inicio do tapete, o
motor ligou!");
}
if (event.getState().equals(PinState.HIGH) && (position != 1) && (foundSwap == 1)) {
	// System.out.print(sdf.format(new Date()));
	// System.out.println(
	// " - Raspberry" + myAgent.getLocalName() + ": Peça no inicio do tapete, o
motor ligou!");

	DFAgentDescription dfd2 = new DFAgentDescription();
	ServiceDescription sd2 = new ServiceDescription();
	dfd2 = new DFAgentDescription();
	sd2 = new ServiceDescription();
	sd2.setType("Raspberry");
	dfd2.addServices(sd2);
try {
	result = DFService.search(myAgent, dfd2);
} catch (FIPAException fe) {
```

```
fe.printStackTrace();
}
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
        for (int i = 0; i < result.length; i++) {
                changeMessage.addReceiver(result[i].getName());
        }
positionSwap = position;
String swapPositionfound = Integer.toString(positionSwap);
System.out.print(sdf.format(new Date()));
System.out.println(" - Raspberry" + myAgent.getLocalName() + " - Swap encontrado
com a position: "+ swapPositionfound);
changeMessage.setProtocol("IAmASwap");
// changeMessage.setContent(swapPositionfound +" " +
// passingSwapPosition);
changeMessage.setContent(swapPositionfound);
myAgent.send(changeMessage);
System.out.println(" - Raspberry" + myAgent.getLocalName() + " - I am a swap.");
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setProtocol("MotorOFF");
changeMessage.setContent("1");
myAgent.send(changeMessage);
}
 else if (event.getState().equals(PinState.HIGH) && (position != 1)) {
runconveyor = 0;
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.addReceiver(conveyorAnterior);
changeMessage.setProtocol("MotorOFF");
changeMessage.setContent("1");
myAgent.send(changeMessage);
}
}
});

outSensor.addListener(new GpioPinListenerDigital() {
public voidhandleGpioPinDigitalStateChangeEvent(GpioPinDigitalStateChangeEvent
event) {
if (event.getState().equals(PinState.HIGH) && (position < NumMaxAgent)) {
        // System.out.print(sdf.format(new Date()));
        // System.out.println(" - Raspberry" +
// myAgent.getLocalName() + ": A minha posicao é: " +// position);
motorOn = 1;
result = null;
DFAgentDescription dfd2 = new DFAgentDescription();
ServiceDescription sd2 = new ServiceDescription();
dfd2 = new DFAgentDescription();
```

```
sd2 = new ServiceDescription();
sd2.setType("Raspberry");
dfd2.addServices(sd2);
try {
        result = DFService.search(myAgent, dfd2);
        } catch (FIPAException fe) {
        fe.printStackTrace();
}

ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
        for (int i = 0; i < result.length; i++) {
                changeMessage.addReceiver(result[i].getName());
                // System.out.print(sdf.format(new Date()));
                // System.out.println(" - Raspberry" +// myAgent.getLocalName() + ":
                Enviei uma mensagem para// "// + result[i].getName().getLocalName());
        }
sentTime = Calendar.getInstance().getTimeInMillis();
String position1 = Integer.toString(position);
changeMessage.setProtocol("MotorOn");
changeMessage.setContent(position1);
myAgent.send(changeMessage);
}
if (event.getState().equals(PinState.HIGH) && (position == NumMaxAgent)) {
        // System.out.print(sdf.format(new Date()));
        // System.out.println(" - Raspberry" +
        // myAgent.getLocalName() + ": A minha posicao é: " +
        // position);
        // DFAgentDescription[] result = null;
DFAgentDescription dfd2 = new DFAgentDescription();
ServiceDescription sd2 = new ServiceDescription();
dfd2 = new DFAgentDescription();
sd2 = new ServiceDescription();
sd2.setType("Raspberry");
dfd2.addServices(sd2);
try {
        result = DFService.search(myAgent, dfd2);
        } catch (FIPAException fe) {
        fe.printStackTrace();
}
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
        for (int i = 0; i < result.length; i++) {
        changeMessage.addReceiver(result[i].getName());
        System.out.print(sdf.format(new Date()));
        }
        sentTime = Calendar.getInstance().getTimeInMillis();
        changeMessage.setProtocol("fim");
```

```
        changeMessage.setContent("1");
        myAgent.send(changeMessage);
        }
if (event.getState().equals(PinState.HIGH) && (position != 1) && (runconveyor== )){
        ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
        changeMessage.setLanguage(codec.getName());
        changeMessage = new ACLMessage(ACLMessage.INFORM);
        changeMessage.addReceiver(conveyorAnterior);
        changeMessage.setProtocol("MotorOFF");
        // String positionSend = Integer.toString(position);
        changeMessage.setContent("1");
        myAgent.send(changeMessage);
                }
        }
});
}
public void action() {
        msgIn = myAgent.receive(mt);
        if (msgIn == null) {
        if (myMotor.isHigh() && (motorOn == 1)) {
        if (waitingTime <= 5000) {
                receiveTime = Calendar.getInstance().getTimeInMillis();
                waitingTime = ((receiveTime - sentTime));
                }
        }
        if (waitingTime > 5000 && waitingTime < 5200) {
                System.out.println(" - waitingTime:" + waitingTime);
                motorOn = 0;
                swapfound = 1;
                waitingTime = 5300;
                if (swapfound == 1) {
                        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
                        System.out.print(sdf.format(new Date()));
                        System.out.println(" - swap encontrado.");
                        // System.out.println(" - WaitingTime:" + waitingTime);
                        DFAgentDescription dfd2 = new DFAgentDescription();
                        ServiceDescription sd2 = new ServiceDescription();
                        dfd2 = new DFAgentDescription();
                        sd2 = new ServiceDescription();
                        sd2.setType("Raspberry");
                        dfd2.addServices(sd2);
                                try {
                                        result = DFService.search(myAgent, dfd2);
                                        } catch (FIPAException fe) {
                                        fe.printStackTrace();
                                        }
        ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
        changeMessage.setLanguage(codec.getName());
        changeMessage = new ACLMessage(ACLMessage.INFORM);
                for (int i = 0; i < result.length; i++) {
```

```
                        changeMessage.addReceiver(result[i].getName());
                }
        passingSwapPosition = position + 1;
        String swapPosition = Integer.toString(passingSwapPosition);
        System.out.print(sdf.format(new Date()));
        System.out.println("-Raspberry"+myAgent.getLocalName()+"-
passingSwapPosition." + passingSwapPosition);
        changeMessage.setProtocol("swapFound");
        changeMessage.setContent(swapPosition);
        myAgent.send(changeMessage);
        swapfound = 0;
        }
        }
} else {
        switch (msgIn.getPerformative()) {
        case ACLMessage.INFORM:
        handleInformMessage(msgIn);
                }
        }
}

public void handleInformMessage(ACLMessage msgIn2) {
ACLMessage changeMessage = new ACLMessage(ACLMessage.INFORM);
SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        if (msgIn2.getProtocol().equals("swapFound")) {
        passingSwapPosition = (Integer.parseInt(msgIn2.getContent()));
        System.out.print(sdf.format(new Date()));
        //System.out.println(" - Raspberry" + myAgent.getLocalName() + " -
passingSwapPosition." + passingSwapPosition);
if (position > passingSwapPosition) {
        waitingTime = 0;
        myMotor.high();
}
if (position == passingSwapPosition) {
        myMotor.low();
}
foundSwap = 1;
System.out.println(" - Raspberry" + myAgent.getLocalName() + " - foundSwap." +
foundSwap);
}
if (msgIn2.getProtocol().equals("IAmASwap")) {
        runconveyor = 1;
        // int positionSwap = Integer.parseInt(msgIn2.getContent().split(" ")[0]);
        positionSwap = Integer.parseInt(msgIn2.getContent());
        //System.out.println(" - Raspberry" + myAgent.getLocalName() + " -
        positionSwap." + positionSwap);
        // int passingSwapPosition =
        // Integer.parseInt(msgIn2.getContent().split(" ")[1]);
//      System.out.println(" - Raspberry" + myAgent.getLocalName() + " -
        passingSwapPosition." + passingSwapPosition);
```

```
if (myposition == positionSwap) {
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - my previous
        positon:" + position);
        myposition = passingSwapPosition;
        position = myposition;
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - my new
        positon:" + position);
} else if (passingSwapPosition == myposition) {
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - my previous
        positon:" + position);
        myposition = positionSwap;
        position = myposition;
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - my new
        positon: " + position);
}
if ((myposition != passingSwapPosition) && (myposition != positionSwap)) {
        myMotor.low();
}
        foundSwap = 0;
}

if (msgIn2.getProtocol().equals("I'm here")){
if (msgIn2.getContent().equals("1")) {
        DFAgentDescription[] result = null;
        DFAgentDescription dfd1;
        ServiceDescription sd1;
        dfd1 = new DFAgentDescription();
        sd1 = new ServiceDescription();
        sd1.setType("Raspberry");
        dfd1.addServices(sd1);
        try{
        result = DFService.search(myAgent, dfd1);
        } catch (
        FIPAException fe)
        {
        fe.printStackTrace();
        }
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
// System.out.print(sdf.format(new Date()));
// System.out.println(" - Raspberry" + myAgent.getLocalName() +
// ": Raspberry encontrados:" + result.length + " agents.");
        for (
        int i = 0; i < result.length; i++)
        {
                changeMessage.addReceiver(result[i].getName());
        }
        NumMaxAgent = result.length;
        System.out.print(sdf.format(new Date()));
```

```
        System.out.println(" - Raspberry" + myAgent.getLocalName() + ":
        Componentes Cyber-Fisicos instalados: "
        + NumMaxAgent);
        }
}
        if (msgIn2.getProtocol().equals("position")) {
        waitingTime = 0;
        int position99 = Integer.parseInt(msgIn2.getContent());
        System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - A minha
        posicao é: " + position);
        if ((position99 == 0) && (inSensor.getState().equals(PinState.LOW)))
        {
        myMotor.low();
        }
        if ((position99 != 0) && (outSensor.getState().equals(PinState.LOW)))
        {
        myMotor.low();
        }
        if ((position99 == 0) && (inSensor.getState().equals(PinState.HIGH))) {
        position = auxposition + 1;
        myposition = position;
        myMotor.high();
        getDataStore().put("myposition", myposition);
        DFAgentDescription[] result = null;
        DFAgentDescription dfd2 = new DFAgentDescription();
        ServiceDescription sd2 = new ServiceDescription();
        dfd2 = new DFAgentDescription();
        sd2 = new ServiceDescription();
        sd2.setType("Raspberry");
        dfd2.addServices(sd2);
        try {
        result = DFService.search(myAgent, dfd2);
        } catch (FIPAException fe) {
        fe.printStackTrace();
        }
        changeMessage.setLanguage(codec.getName());
        changeMessage = new ACLMessage(ACLMessage.INFORM);
for (int i = 0; i < result.length; i++) {
        changeMessage.addReceiver(result[i].getName());
}
String position1 = Integer.toString(position);
changeMessage.setProtocol("updateposition");
changeMessage.setContent(position1);
myAgent.send(changeMessage);
}
}
if (msgIn2.getProtocol().equals("updateposition") &&
(!msgIn2.getSender().getName().equals(myAgent.getName())
&& (position >= Integer.parseInt(msgIn2.getContent())))) {
```

```
position = position + 1;
}
if (msgIn2.getProtocol().equals("MotorOn")) {
        int position77 = Integer.parseInt(msgIn2.getContent());
        auxposition = position77;
        waitingTime = 0;
        if (position == 0 || (position == auxposition + 1)) {
        myMotor.high();
        conveyorAnterior = msgIn.getSender();
}
if (position == auxposition && auxposition != 0) {
        myMotor.high();
}
}
if (msgIn2.getProtocol().equals("positionOut")){
        int positionout = Integer.parseInt(msgIn2.getContent());
        // System.out.println(" - Raspberry" + myAgent.getLocalName() +
        // "positionout:" + positionout);
        if (positionout < position)
        {
                position = position - 1;
                myposition = position;
                getDataStore().put("myposition", myposition);
                System.out.print(sdf.format(new Date()));
                System.out.println(" - Raspberry" + myAgent.getLocalName() + "A
                minha posição: " + myposition);
        }
DFAgentDescription[] result = null;
DFAgentDescription dfd1;
ServiceDescription sd1;
dfd1 = new DFAgentDescription();
sd1 = new ServiceDescription();
sd1.setType("Raspberry");
dfd1.addServices(sd1);
try{
        result = DFService.search(myAgent, dfd1);
        } catch (
        FIPAException fe)
        {
        fe.printStackTrace();
}
changeMessage.setLanguage(codec.getName());
changeMessage = new ACLMessage(ACLMessage.INFORM);
// System.out.print(sdf.format(new Date()));
// System.out.println( " - Raspberry" + myAgent.getLocalName() + ": Raspberry
encontrados:" + result.length + " agents.");
        for (
                int i = 0; i < result.length; i++)
                {
                changeMessage.addReceiver(result[i].getName());
```

```
                // System.out.println(" - Raspberry" +
                // myAgent.getLocalName() +
                // ": Enviei uma mensagem para "+result[i].getName().getLocalName());
        }
NumMaxAgent2 = result.length;
if (NumMaxAgent != NumMaxAgent2)
{
        NumMaxAgent = NumMaxAgent2;
        System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + ":
        Componentes Cyber-Fisicos instalados: "+ NumMaxAgent);
}
}
if (msgIn2.getProtocol().equals("fim"))
{
        if (msgIn2.getContent().equals("1")) {
                if (position == NumMaxAgent) {
                myMotor.low();
                System.out.print(sdf.format(new Date()));
                System.out.println(" - Raspberry" + myAgent.getLocalName() + "A
                minha posição: " + myposition);
                }
        }
}
if (msgIn2.getProtocol().equals("MotorOFF"))
{
        if (msgIn2.getContent().equals("1")) {
                myMotor.low();
                waitingTime = 0;
                motorOn = 0;
        }
}
if (msgIn2.getProtocol().equals("swapFoundconveyor1")) {
        // if ((position != passingSwapPosition) && (position != 1)) {
        // myMotor.low();
        // }
        // if (position == 1) {
                // position = Integer.parseInt(msgIn2.getContent());
                // System.out.println(" - Raspberry" + myAgent.getLocalName() + " -
                // position." + position);
                // passingSwapPosition = 0;
                // } else if (position == Integer.parseInt(msgIn2.getContent())) {
                // position = 1;
                // System.out.println(" - Raspberry" + myAgent.getLocalName() + " -
                // position." + position);
                // myMotor.high();
                // passingSwapPosition = 0;
                // }

        if (myposition == 1) {
```

```
        // System.out.print(sdf.format(new Date()));
        // System.out
        // .println(" - Raspberry" + myAgent.getLocalName() + " - A
        // minha posicao anterior = " + position);
        myposition = Integer.parseInt(msgIn2.getContent());
        position = myposition;
        System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - A minha
        posicao: " + position);
        } else if (Integer.parseInt(msgIn2.getContent()) == myposition) {
        // System.out.print(sdf.format(new Date()));
        // System.out.println(" - Raspberry" + myAgent.getLocalName() + " - A
        // minha posicao anterior = " + position);
        myposition = 1;
        position = myposition;
        System.out.print(sdf.format(new Date()));
        System.out.println(" - Raspberry" + myAgent.getLocalName() + " - A minha
        posicao: " + position);
        }
if ((myposition != Integer.parseInt(msgIn2.getContent())) && (myposition > 1)) {
        myMotor.low();
}
}
}
        @Override
        public boolean done() {
        // TODO Auto-generated method stub
        return false;
}
}
```