**基于多Agent的动态调度系统的设计与研究**

**1 绪论**

**1.1 研究背景和意义**

当今制造业正经历深层次的变革，生产模式正逐渐从从“小品种，大批量”到“多品种，小批量”的转变，这是由于科学技术的快速发展，人们生活水平有了极大的提高，人们的生活需求不再局限于基本的物质生活保障，而是追求商品的个性化。随着市场竞争的日益激烈，产品个性化使得现代制造系统的运行环境越来越充满随机性：不可预知的任务到来或撤销、制造资源的紧缺故障等等，诸多不确定性、动态性和复杂性使得生产过程中的作业调度问题更加难以解决。生产调度是制造系统的关键技术。现代制造对新一代调度系统提出了新的要求，除了要求其具有稳定性、可靠性、可维护性等传统调度系统应有的特性外，还要具有可重构性、敏捷性和适应性等特点。

要解决现代制造中复杂的调度问题，仅仅依靠研究调度算法是不足够的，还需要从系统的角度出发，研究理想的调度系统模型以及合理的调度系统结构。传统的调度系统采用集中式控制结构，但这种结构应用在问题规模庞大的复杂系统时，呈现出可扩展性、容错性差以及市场响应能力低下的特点，这种控制结构现在正逐渐退出市场。随着计算机技术的飞速发展，车间控制结构趋向于将系统职能进行分化，即各控制实体各司其职。同时，为了提高系统的动态适应性、容错性以及敏捷性，其发展趋势是越来越强调控制实体的自治性和相互间的协调能力。

近年来，根据对现代制造系统和分布式人工智能的研究表明，由具有自主性、社交性和能动性的Agent彼此相互协作组成的分布式制造系统具有很高的灵活性，可以快速响应环境的动态变化。多Agent系统降低了集中控制的限制，提高了分散控制和并行处理的能力。这样，制造系统就可以通过分布在车间的资源和Agent之间的联合来实现系统的集成。

现阶段，关于多Agent的完整的理论体系结构尚未成形和普及，基于Agent系统应用的成功范例并不多见。考虑现代车间调度问题的复杂性，显然是典型的NP难问题，因此至今为止的研究成果依然未能彻底解决生产调度在实际应用中所面临的诸多难题。我们希望能够开辟一种新的思路，集成Agent和调度策略等先进的技术和理论，设计一种实用、低成本，又能快速响应市场的调度机制，以达到优化制造业控制模式，提高企业竞争力的目的。

项目的工作主要集中在以下两方面：

1. 制造车间动态生产调度中多Agent系统的研究，工作包括多Agent系统建模，系统结构的构建以及系统中各Agent之间的交互协作机制。
2. 建立基于多Agent的动态调度原型系统，工作包括设计基于Java面向对象技术和Socket通信机制的Agent通信模型以及在原型系统的基础上对常规任务调度和异常调度进行验证以及可行性分析。

**1.2 动态调度问题的研究现状**

在上世纪七八十年代，研究人员的研究重心主要放在传统的车间调度问题，也就是面向静态的长期的调度任务，由于问题规模通常比较小，任务生产周期的时间跨度很长，因此响应的计算量也较小，通常采用的方法主要是精确算法，如枚举法，分支界定法等。随着计算机技术和人工智能的飞速发展，也随着对一些生物自然规律的认知加深，研究人员逐渐提出了更多行之有效的优化算法，这些算法较好地克服了传统算法的局限性，更加适用于现代制造业的车间调度问题，这些算法包括，遗传算法、蚁群算法、粒子群算法、模拟退火算法、禁忌算法等，

随着客户需求的个性化，再加上生产加工过程中加入了更多的不确定性和更复杂的约束，这使得上述调度方法并不能完全适应于企业的敏捷化制造的要求。而具有良好灵活性、社会性和合作能力的多Agent调度系统提供了应急和快速问题求解的能力。近年来，不断有研究人员将多Agent技术应用于车间调度的问题研究上，并且已经取得了一定的成果。例如，华中理工大学以数控加工系统为背景，开发了一个基于多Agent的分布式网络化调度原型系统，实现了制造单元的柔性智能化。沈阳自动化所把多Agent系统应用在多吉携手的协调与协作过程中，并取得阶段性成果。此外，文献[1]提出了一种基于现场总线的多Agent作业车间动态调度模型，首先由一个主Agent向其他辅助Agent发送调度任务，通过对回收的所有调度结果进行统一判断和协调，最终得到一个可行的调度方案。文献[8]为提高调度系统对生产环境经常发生变化的自适应能力和全局优化能力,提出了一种基于蚂蚁智能与强化学习相结合的协商策略的多Agent动态调度方法。文献[9]为实现在保持高生产率的前提下对环境变化的快速响应 ,提出了一个基于自适应Agent的模型 ,研究了制造系统中最常见的任务在加工资源上的分配问题。同时还设计了在多智能体系统中分别代表加工资源、任务和协调者的三类Agent ,以及它们在各自利益驱动下所应遵循的行为规则。文献[10]探究各种结构的多Agent系统并分析了它们的优劣。同时还研究了当前应用在多Agent系统的三种主流算法，，dispatching、Scheduling和Pull算法，根据平衡性、扩展性和健壮性三个指标分析了三种算法的适用场景，为多Agent系统的构建和应用提供了很好的指导。文献[11]从制造系统的需求和发展趋势出发，研究基于多Agent的制造车间动态调度问题，通过建立一个基于多Agent的制造车间动态调度原型系统，对Agent理论与技术在智能制造中的应用进行了探索性研究。作者还提出了基于“规则协商”的调度机制和调度策略。将基于规则的调度方法与Agent之间的协商机制相集成，充分发挥各Agent的智能性、自治性和协调能力。文献[12]提出一种短期规划与动态响应能力相结合的混合协商策略。该策略针对柔性车间生产调度问题和面对扰动的再调度问题,将组合拍卖与改进合同网相结合,通过规定相关智能体的动作和交互顺序,实现系统在平稳或扰动状态下的生产调度。基于上述协商策略,开发了一个基于智能体的生产调度原型系统,并通过仿真实验验证协商策略的可行性和有效性。文献[13]为解决制造系统的敏捷生产调度问题，先通过基于功能分解的方法,给出了管理、资源和工件等三类Agent基本组件组成的分布式多Agent调度系统结构、Agent组件基本结构及定义.然后利用基于分布式规则的方法,建立了Agent间的协调策略和调度机制,实现了敏捷生产调度。

**2 基于多Agent的动态调度系统结构的分析和设计**

我们的多Agent系统的体系结构主要是基于层次型模型进行构建的，即Agent之间存在多级主从关系。每一层Agent是上一层Agent的从Agent，同时又是下一层Agent的主Agent。下层Agent的行为受到上层Agent的制约。Agent间的交互协作是基于合同网的“招标——投标——中标”机制。在任务的动态分配过程中，上层主Agent定义子问题合同后，能够向下层从Agent发起投标。下层Agent得到子问题合同后，可以进一步对问题进行分解并向下一层Agent发起投标。当有Agent结点能够满足合同要求并承诺具有完全负责问题求解的能力，就可以向主Agent确认投标。当有多个从Agent投标，主Agent就会对各个从Agent的投标书进行评定，从中选取成绩最优的投标书，然后通过相关的通信接口下达确认消息到从Agent，从Agent接收合同后，就可以进入问题求解的过程。

**2.1 基于多Agent的动态调度系统建模**

多Agent系统由多个分布和并行工作的Agent通过协作完成某些任务或达到某些目标的计算系统。单个Agent的能力是有限的，但可以通过适当的体系结构把Agent组织起来,从而使整个系统能力超过任何单个Agent的能力。这样的系统具有传统分布，并发问题求解的优点,同时具有复杂的交互模式。

制造系统的生产调度问题是随问题规模呈指数成倍增长的NP难型。实际生产中影响系统变化的因素较多，如生产种类的多样性、零件种类的多样性、制造工艺的多样性、环境条件的不确定性等，而这种制造系统的复杂性伴随在生产调度的整个过程之中。我们的目标正是以Agent模型及多Agent体系结构来研究生产加工过程的优化调度问题，用多Agent技术来处理调度过程的复杂性，选取合理的资源分配方案,更好地满足市场对柔性制造的要求，因此Agent的内部模型和系统内部结构的合理性将对系统求解问题的效率和稳定性有着至关重要的影响。

根据现代制造系统的问题规模以及任务特点，我们定义了五类Agent作为基于多Agent的动态调度系统的基本组成构件，包括管理Agent、资源Agent、工艺Agent、算法Agent和监控Agent。



管理Agent：相当于一个虚拟的车间管理员。主要负责生产订单任务的输入和分配以及对整个多Agent系统运行状况的监控和日志文件的记录。

资源Agent：表征制造系统的生产设备。每个资源Agent负责其所属范围内设备的管理。单独的生产设备、制造车间以及完整的设备集合均有对应的资源Agent。资源Agent间的层次模型构成了多Agent系统的基础架构。资源Agent负责对任务进行分解，生成子任务合同并向下层从Agent发起招标。各资源Agent投标过程需要相互竞争以获得生产任务。

工艺Agent：表征制造系统所能生产的产品。一个产品由若干个工件组成，一个工件的成型需要经过若干道工序的加工。工艺Agent就负责记录各产品信息以及相应的工件加工流程图，同时提供数据咨询接口。

算法Agent：根据设备集合以及生产任务进行问题求解。资源Agent负责对生产任务进行分解以得出若干可行解，每个可行解对应一个设备集合，算法Agent根据集合内设备的工作状况以及生产任务得出最优的调度结果。

监控Agent：表征制造系统的监管人员，负责监控生产线信息的采集以及故障分析。监控Agent采集生产线各个工位的工作状态信息，根据信息分析工位是处于安全状态还是故障状态，并把相关信息反馈到管理Agent，由管理Agent判断调度方案的重构。

动态调度系统的工作流程如下：

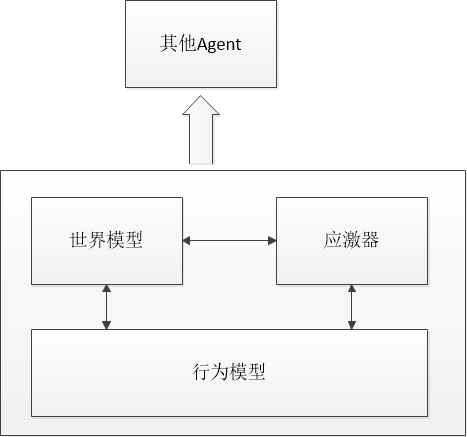
1. 调度系统运行过程中，五类Agent之间进行对等的通信，若干资源Agent间采用层次广播通信。当输入生产任务时，管理Agent根据自身数据库的信息对任务进行合法性判断，例如需要判定系统是否有能力生产该产品、系统生产原料能否支撑该任务的执行、任务的完成期限是否合理等等。当合法性通过后，计算任务的优先级，并根据优先级放置到任务队列的相应位置，并向资源Agent发布任务。当资源Agent正在进行设备的管理和调度时，管理Agent将保存和维持任务队列，直到资源Agent空闲。
2. 资源Agent获取加工任务后，向工艺Agent请求数据咨询，获取产品信息、工件的工艺流程图等等，之后就能采用招标——投标——中标的策略进行任务的动态分配。首先，上层Agent根据工艺Agent返回的产品/工件信息对任务进行分解，生成子问题合同并向下层Agent发起招标，经过规定的招标时间后，资源Agent对下层Agent的标值进行比较，选取标值最小的Agent中标。最后加工任务就在中标的Agent对应的生产设备上进行加工。
3. 在上一步资源Agent的招标/投标过程中，Agent在招标中是根据自己所管辖的设备集合以及招标的子任务集合，向算法Agent发起请求。算法Agent根据设备集合中各设备的工作状态和加工信息，以最小化最大完成时间为目标生成最优的调度方案，并将完成时间作为标值返回给资源Agent。
4. 确定中标对象后，资源Agent把调度结果返回给管理Agent并录入加工日志。之后

资源Agent不断检查管理Agent的任务队列，当有新任务后就重复上述过程，需要指明的是，监控Agent扮演的是监管的角色，并不直接参与调度过程，而是在调度系统的整个运行过程中负责对生产线进行数据的采集和故障分析。当故障发生时，由管理Agent判定是否进行重调度。同时，由于资源Agent中记录并模拟了实际生产线各设备的工作状态，如设备的工序队列，各工序的开始加工时间以及完成时间等等，因此虚拟生产线和实际生产线的时间误差观测就交给监控Agent，当两者的时间误差超过规定值，就需要通知管理Agent对各资源Agent进行调整。

从上述的简单描述可以看出，五类Agent之间是一种平等，并行的关系。多Agent动态调度系统的分布式结构保证了系统的灵活性和可扩展性，各类Agent之间的交互保证了系统的自治性、智能性和社会性。五类Agent中，资源Agent的数量是最庞大的，如果资源Agent之间也采用对等的通信模型，那么在任务的招标和投标过程中，由于各Agent都能参与投标，势必会导致频繁的通信，占据大量的通信资源，降低系统的运行效率。对资源Agent采用分层次的结构后，在同一时间，任务的招标、投标过程只在层与层之间发生，同时由于对任务进行了分解，是以工序的集合为单位进行招标投标，而不是以单个工序为单位，这样能显著减少通信的冗余，提高系统的稳定性。

**2.2 各类Agent内部模型的建立**

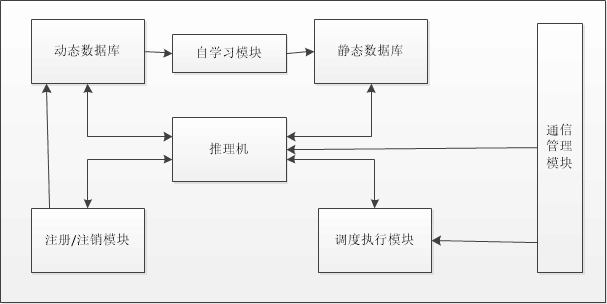
Agent的内部模型与多Agent系统的体系结构有着密切的联系，前者的工作效率和性能对整个系统结构和性能有着至关重要的影响。我们的Agent可以抽象为三个部分：世界模型、应激器和行为模型。其中世界模型包含了Agent的外部环境以及自身的描述信息。应激器负责对外界的刺激行为和信息做出反应。行为模型根据应激器的反应以及世界模型的信息，针对要达到的目标生成相应的规划以及采取相应的行为。



Agent的抽象模型

在动态调度系统建模中，每个Agent都可以表示具有一定智能和自治性的物理实体或者逻辑实体，例如资源Agent表示生产设备，工艺Agent表示产品和工件信息，算法Agent表示调度逻辑等等。尽管各类型Agent的功能各不相同，但是我们根据多Agent动态调度系统基本运行过程和特性的基础上，提取出Agent内部模型的基本结构，根据功能的不同，会有少许的差异。Agent间的差异体现在各模块的实现上，它们通过一定的交互协作机制来解决复杂的制造问题，最终给出一个合理的调度结果。

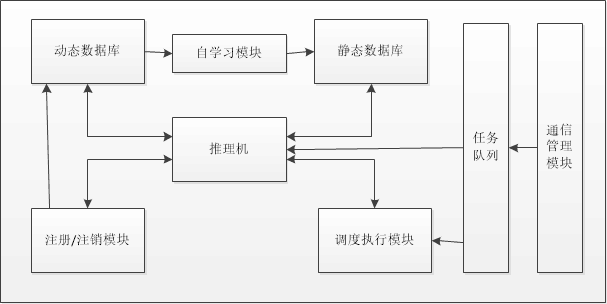
基于多Agent的动态调度系统内Agent的内部模型的基本结构如下，包括动态数据库，静态数据库，自学习模块，推理机，调度执行模块以及通讯调度模块。各模块根据不同Agent在功能上会存在差异。



Agent内部模型基本你结构

**2.2.1 管理Agent的内部模型**

管理Agent负责生产订单任务的输入和分配以及对整个多Agent系统运行状况的监控和日志文件的记录，因此我们对其内部结构设计如下：



管理Agent内部模型

1. 动态数据库：记录当前运行的Agent的注册信息，包括Agent的名字、能力、状态等。还记录任务队列各任务的状态，包括任务的数量，任务的优先级等。
2. 静态数据库：包括部分工艺信息，加工日志文件，与常规调度和异常调度策略相关的参数等，这些参数能影响资源Agent任务调度和异常调度的结果。
3. 自学习模块：运行期间负责对动态数据库中的信息进行筛选，从中选取必要有价值的信息添加到静态数据库中。例如当有新产品能投入生产时，动态数据库需要把改信息添加到静态数据库。
4. 注册/注销模块：当有新的Agent添加到系统时，需要把Agent的信息添加到动态数据库，以作为管理Agent监测其它Agent运行状况的依据。
5. 推理机：管理Agent的推理机负责对订单任务的合法性进行判断以及对其它Agent的状态进行监测。例如：推理机在系统通信量较低时向注册的每个Agent发送探询包，不可达的Agent将视为发生故障，注销Agent并通知管理员；当某些Agent陷入死循环或若干Agent之间形成死锁时，也需要推理机破坏死锁状态。
6. 调度执行模块：负责对任务队列进行操作，例如每隔特定时间查询动态数据库看，重新计算各任务的优先级并对任务进行排序。查询和任务排序过程均不需要人工干预。
7. 通信管理模块：为管理Agent与外界通信提供接口，该接口是基于Socket通信机制实现的。

**2.2.2 资源Agent的内部模型**

资源Agent表征制造系统的生产设备，负责对任务进行分解并执行招标、投标过程。其内部模型设计与管理Agent相同。

1. 动态数据库：记录上层Agent发布的任务合同书，表征设备的工作状态（工序序列，各工序的开始加工时间和完成时间），其下层Agent的状态等等。
2. 静态数据库：记录表征设备的能力，例如可加工的工序以及加工时间；记录其下层Agent的信息，如Agent的名字、能力等；记录与任务分解、合同书投标相关的策略参数。
3. 自学习模块：从动态数据库中获取必要数据并对静态数据库进行修改，例如当有新的资源Agent注册为子Agent时，需要将其信息添加到静态数据库；例如当设备发生局部故障工作效率下降时，需要根据实际情况修改设备的生产能力。
4. 注册/注销模块：用于向管理Agent注册或注销自身的信息。
5. 推理机：主要进行与招标相关的任务分解工作以及和投标相关的推理计算。首先接到竞标通知，并根据自身当前状态、投标策略以及算法Agent的计算得到标值，并返回给上层Agent。
6. 调度执行模块：根据推理机任务分解的结果以及动态数据库中设备的工作状态与算法Agent进行交互。
7. 通信管理模块：与管理Agent的通信管理模块功能相同。

**2.2.3 工艺Agent的内部模型**

表征制造系统所能生产的产品。工艺Agent的内部模型与Agent内部模型基本结构相同，其中略去通信管理模块和注册/注销模块的说明。

1. 动态数据库：记录工艺Agent运行期间接收到的关于产品信息增加或删改的信息。
2. 静态数据库：记录系统所能生产的产品信息，包括产品的工件组成，工件的工艺流程等。
3. 自学习模块：将动态数据库中关于产品信息的增加或删改对静态数据库进行操作。
4. 调度执行模块：为资源Agent和管理Agent提供数据查询接口。

2.2.4 算法Agent的内部模型

算法Agent根据设备集合以及生产任务进行问题求解，是动态调度的核心。算法Agent的内部模型如下，注册/注销模块以及通信管理模块略去。

1. 动态数据库：记录缓存运行期间所查询到的与工序加工信息相关的数据。为了降低通信量，提高工作效率，当动态数据库中存在所需工序信息的缓存时，若缓存的保存时间未超过规定的时间，可以直接使用。其次动态数据库记录与调度算法相关的调度参数的修改。
2. 静态数据库：主要保存与调度算法的计算推理流程以及与算法相关的参数。
3. 自学习模块：为获取更令人满意的调度结果，需要每隔固定时间就对算法进行调整，因此需要从动态数据库获取对算法参数相关的修改。
4. 调度模块：负责接收来自关联的资源Agent的请求，获取设备集合以及任务集合。根据任务集合中与工序相关的信息向工艺Agent发起数据请求。最后把调度方案的完成时间作为标值返回到关联的资源Agent。
5. 推理机：主要进行调度方案的生成。推理机根据设备集合和若干道工件工序，按照调度算法给出最优的调度方案，即决定各个工序应当安排在设备集合中哪一台设备，以及应当在哪个时刻开始加工才能使得完成时间尽可能小。最终的完成时间作为投标的标值。

**2.2.5 监控Agent的内部模型**

监控Agent负责监控生产线信息的采集以及故障分析，其内部模型如下，通信管理模块和注册/注销模块略去。

1. 动态数据库：记录当前生产线上各个工位的工作状态，如已完成的操作、工作状态（工作中、暂停中、已故障）、设备的加工状态（如当前正在加工的工序以及代加工的工序）等等。
2. 静态数据库：记录生产线工位数据的历史数据。
3. 自学习模块：定时将动态数据库中的数据写入静态数据库。
4. 推理机：主要负责对动态数据库中的数据进行故障分析以及时间同步。当设备故障时，需要分析故障的类型、程度以及对工作效率的影响。此外还需要对资源Agent模拟的生产线与实际生产线的时间误差进行判定。
5. 执行调度模块：负责接收来自生产线的工位数据以及反馈故障信息以及时间误差到管理Agent。

**3 动态调度系统中Agent的交互协调机制**

在基于多Agent的动态调度系统中，各个Agent既相互独立又彼此联系，依靠自身的自治能力以及各方的合作能力共同完成系统的调度目标。在调度过程中，调度方案的生成以及加工任务的完成都依赖于Agent之间良好的协调与合作。因此合理高效的交互协调机制是Agent有效完成调度任务的基础，不仅可以实现Agent的智能性，还能保证Agent之间进行高效的合作。

**3.1 资源Agent间的主从交互协调机制**

我们的动态调度系统中相当一部分的通信量是集中在资源Agent之间的交互。这是由于调度问题的求解主要在于任务的分解，投标和竞标过程中，随着问题规模和生产设备规模的增长，必然导致求解需要的运算量急剧增加，而这些过程都是由资源Agent和算法Agent之间的交互协调合作共同完成的，因此减少Agent之间的通信量能有效提高系统对市场和环境的快速应答能力。多Agent系统是基于分布式的系统，但是由于分布式结构较为松散，提高系统灵活性的同时却也导致了整体优化能力较低的弊端。例如，当资源Agent的规模大幅增长时，仅仅依赖各Agent的自主性来相互协调解决问题时，若Agent之间对资源和任务的动态分配产生分歧甚至冲突时，可能会导致系统的死锁。因此我们设计了主从交互协调机制来解决这类问题。

“主从”代表资源Agent之间的分层结构。上层Agent对下层Agent具有约束关系，下层Agent需要遵从上层Agent的决策。当Agent之间的行为相互冲突产生分歧时，要以上级Agent的决策为先，防止系统死锁，尽可能做到系统目标和个体Agent目标之间的协调，利于全局目标的优化。在这里管理Agent对系统内其它Agent具有指导作用，即管理Agent可看作是所有Agent的上级Agent。其它Agent的某些决策需要通过管理Agent的审核后方可执行。例如当系统发生设备故障或是任务的变更撤销，是否需要进行重调度需要经由管理Agent判断。另外当其他Agent遇到彼此协商难以解决的问题时，也会由管理Agent来协调，例如当有优先级更高的任务到来，某个资源Agent需要把后续加工的任务转交到别的资源Agent时，而目标资源Agent同时也在加工高优先级的任务无法承担额外任务时，就需要管理Agent的协调工作，否则系统可能会出现死锁，这是管理Agent可以通过回收任务，调整任务优先级等措施。

资源Agent间任务的分解和分派是基于合同网的协商机制。当一个没有能力完成某一任务时，它便可以通过之间的协商将任务分配给其他完成，这样，任务分配机制将直接影响到多系统的性能。传统的合同网协商过程中，由一个管理者负责分配任务，在这个管理者中存储有在当前系统中各结点的能力表。当有任务需要完成时，就根据当前状态表询问有此能力的是否接受此任务，如果接收到同意的标书，会告诉管理者，否则告知当前没有可以完成此任务。传统合同网的主要缺点是非常容易消耗大量的通信带宽，当系统中的增加时，所要管理的消息是数目的平方，因此极易引起通信间的瓶颈反应。其次由于所有的结点都可以参加投标,这就要求通信频繁和丰富的资源。管理器必须评价大量的投标书，使用大量的资源和耗费大量的时间。公共的投标势必造成管理器负载过重，甚至可能导致系统的死锁。而且，每个结点只是被动地接收任务，除却管理者外，其他结点之间信息共享程度低，灵活性差。

另外，传统合同网中管理器在指派任务合同后要等待接受有关结点的信息，当确认信息到来之前，管理器不知道结点是否接受合同。如果没有结点参与投标，管理器势必陷入无限制的等待中，也容易引发饥饿现象或者造成系统的死锁。值得注意的一点是，由于结点竞标过于自由，因此竞标过程可能耗费过长的时间，这会延长问题的求解时间和降低系统处理问题的速度,难以保证结果最优。

在我们的资源Agent调度过程中，我们只允许信息在层次间进行传递。例如上层主Agent发布的任务合同，只允许下层从Agent进行竞标。也就是说，在我们的系统中并不要求所有的问题都经过所有的Agent进行交互。系统通过各个层次之间灵活地协调与合作，可在满足系统全局性能要求的前提下，最大限度地发挥各自的灵活性和自主决策能力，提高了对急件、机器故障等意外事件的反应、处理能力，满足敏捷制造环境下动态调度的要求。

资源Agent的交互协调过程大致包含任务的分解和发布、竞标和任务分配等。

任务的分解和发布：当资源Agent从管理Agent的任务队列中获取优先级最高的任务后，根据自身的分解策略对任务进行分解，得到子问题合同，然后采用广播方式，对下层Agent进行合同书的发布，合同书中注明任务的截止时间，任务的性质、数量等属性。

竞标：下层Agent收到任务合同书后，生成匹配任务要求的设备集合，连同任务信息传达到对应的算法Agent。算法Agent根据自身的调度策略得到最优的调度方案，其中完成时间作为标值。

任务分配：在规定的竞标时间内，上层资源Agent根据最小标值选择最佳资源Agent中标。并通知该Agent，得到合同确认后，指定其完成任务。对于竞标失败的Agent，根据标值的大小从中选择若干调度方案作为备选方案保存在管理Agent中，当突发状况发生，如设备故障时，若备选方案中设计的设备工作状态没有改变，或者改变较小不造成冲突时，可以直接使用备选方案，这样可以避免重调度的时间消耗，保证系统的实时性和高效性。

**3.2 对等交互协调机制**

在我们的调度系统中，管理Agent接收生产任务后，与资源Agent协商，确定作业计划，完成对系统生产资源、设备的动态分配和管理，并接收来自监控Agent的反馈信息，实现对生产线信息的采集和故障分析，达到生产过程的监控个目的。资源Agent根据加工任务的性质，与算法Agent交互合作，得出最优的生产计划。

在以上动态调度过程中，各Agent就某一项生产订单任务往往需要和其它Agent间进行协作，这时候采用的是对等交互协调机制。五类Agent中，除了管理Agent作为上级Agent外，其它各类Agent，如资源Agent、算法Agent、工艺Agent以及监控Agent之间的交互是使用对等通信模式，即它们之间并没有主从关系。对等通信过程中，实际是Agent根据自己的需求寻求最佳合作对象的过程。Agent为了完成自身的任务，满足个体的需要，从行为模型中选择特定的行为，当其他Agent接收到应激信息后，同样采取特定的行为作出应答。例如当加工某道工序的过程中，设备发生了故障，为了避免大规模的重调度，故障设备对应的资源Agent就可以和其它资源Agent进行交互，需求能加工此工序的资源Agent，请求合作。若存在资源Agent给予应答，就可以把工件运输到指定的资源Agent，继续未完成的工作。这需要管理Agent维持一个Agent与Agent之间的对应关系，例如将具有同样加工能力的设备作为一个集合，集合内的设备只向集合内的其它资源Agent发起合作，这样可以避免盲目地对全部Agent进行查询。

**4 基于多Agent动态调度原型系统的设计和开发**

目前面向多Aent的系统开发尚处于初步阶段，开发方法并不成熟，这主要是由于Agent的开发语言ACL跟当前主流的程序设计语言不同，它还处于开发和推广的初级阶段，对API的设计还不够完善，因此开发人员在面向Agent开发中面临许多问题和困难，己出现的一些开发方法和平台缺乏系统性和标准化，面向Agent的开发工作存在一些随意性。从这一点来讲,多Agent更多的知识提供一个面向制造系统应用的功能模型，而如何构造基于多Agent的执行系统则是我们需要解决的一个关键问题。在我们的工作中，为了解决多Agent系统的实现问题，我们使用面向对象技术、Java语言和Socket通信机制来研究基于多Agent的动态调度原型系统的设计与实现问题。

**4.1 面向Agent的开发语言**

Agent可以表示一个实际物理对象或逻辑对象，它能自主地执行给定的任务。因此,可以用它作为分析和设计制造系统的模型对象。从我们之前对它概念、组成和内部模型来看，Agent在封装性、继承性和多态性等方面与面向对象方法中的对象有很多共同之处，所以我们可以采用面向对象的分析与设计方法来分析和建立Agent的模型，使我们构建对象具有Agent的特性，即自治性、只能性和对环境的反应性、交互性等能力。比如,可以将Agein描述成对象类，可以通过消息传递实现交互。其自主性、主动性，可以通过事件驱动方式，通过一些程序、算法来实现。其内部具有的自适应、自组织的能力可以利用建立数据库以及对象的筛选机制来实现。于是在应用面向对象技术实现Agent的过程中，可将面向对象方法作为实现Agent实体的工具。

我们选择Java来设计基于多Agent的动态调度原型系统，这是由于Java语言独特的分布式、面向对象和跨平台的特性，与基于多Agent系统的分布式协调合作机制是相吻合的。而且，Java语言的设计完全是面向对象的，Java支持静态和动态风格的代码继承及重用。Java语言集中于对象及其接口，它提供了简单的类机制以及动态的接口模型。同时，Java同C/C++相比，安全性、封装性更好(如对全局变量，而在C/C++中，依赖不加封装的全局变量常常造成系统的崩溃)，更容易让用户掌握。同时，Java允许程序动态地装入运行过程中所需要地类，这是C++语言进行面向对象程序设计时无法实现的。而且，Java通过接口来支持多重继承，使之比严格的类继承具有更灵活的方式和扩展性。Java中的异常处理机制也使程序更可读且更结构化。

**4.2 原型系统的设计**

基于多Agent的动态调度原型系统实现的关键在于Agent之间的通讯和合作问题。在系统实现过程中，将系统框架如下建立：以Java作为Agent开发语言、以Java语言中的Socket通信、多线程的流传输的方法作为Agent通讯和互操作的通讯模型。下图描绘了原型系统的框架结构。



其中，上层是控制层，实现Agent之间的合作控制，体现了Agent之间的交互和协作策略，以及整个多Agent系统的体系结构及多Agent之间的协调、合作方法。下层是通讯层,实现Agent之间的信息交换与交互细节。例如合同网协商机制属于上层，知识查询和时间处理属于下层。为了实现基于多Agent动态调度原型系统，主要应从两个方面着手，一是Agent的行为模型的实现，即软件结构；二是Agent之间的通讯与合作。需要建立不同Agent的通讯模型，实现Agent之间的协作与交互。

的车间调度原型系统。

**4.3 原型系统的架构**

基于多Agent的动态调度原型系统可以分为三大子系统以及两个模块，分别是管理子系统、资源子系统、监控子系统以及算法调度模块和工艺管理模块。系统各部分之间使用Socket通信、多线程的传输流的方法来解决Agent通讯和互操作的问题。系统架构如下图所示：



我们的原型系统是基于分布式的多Agent系统，目的是能够将该系统应用在分布式环境中，例如多机系统。在分布式环境中，每个Agent可以表征一个物理实体或者软件逻辑实体，例如可以表示一台设备，这就是我们系统中的资源Agent；也可以把Agent对应为一个过程，例如我们将调度算法封装到Agent中，形成我们的算法Agent。分布式环境中，每个Agent可以看做一个单独的进程，每个进程有自己的工作空间，互不干扰。为了提高每个Agent的工作效率，增强灵活性，为每个Agent创建若干个线程，每个线程可用于处理多个请求，进行特定的计算、调度等。

**4.3.1 管理Agent和资源Agent的通信实现**

管理Agent是整个原型系统的核心，负责管理、监控和协调其他Agent的工作。管理Agent还负责接收外界的订单任务，订单任务可根据用户的需求生成特定格式的消息或者数据包，经HTTP/POST方式传送到管理Agent，也可以由管理员直接下达订单任务。管理Agent中有特定的线程专门用于接收外界的订单任务，同时还专门开辟一块特定大小的缓冲区用于接收订单任务的数据包，这块缓冲区就是任务队列。根据管理Agent所在的计算机的请求并发处理能力的大小，我们可以为缓冲区设置不同的大小，这是为了避免当用户订单过多造成管理Agent负载过重，严重影响原型系统整体的运行效率。如计算机的处理速度较快和内存容量较大，即能快速处理多个请求，我们可以将缓冲区设置稍得大些。当任务队列不为空，资源Agent从中逐个取出任务并进行处理；当任务队列为空时，资源Agent的线程处于休眠状态直至有新的任务到达任务队列，如下图所示：



我们为任务队列设置两个条件变量Condition，分别为notFull和notEmpty。当任务队列为空时，资源Agent的任务获取线程使用notEmpty.await阻塞从而陷入睡眠。当管理Agent的任务接收线程获取到任务订单并投放到任务队列时，使用notEmpty.signal唤醒沉睡的资源Agent线程，通知该线程获取并处理任务；同理，当任务队列满时，管理Agent的任务接收线程使用notFull.await阻塞从而陷入睡眠。当资源Agent从任务队列取出任务后，使用notFull.signal唤醒任务接收线程，通知该线程把新任务投放到任务队列中。使用线程间的同步机制能保证资源Agent和管理Agent之间的通信的有序和高效，当缓冲区为空或或满时，对应的线程陷入睡眠释放处理器，减少系统资源的负载。

**4.3.2资源Agent间的通信实现**

多个资源Agent组成原型系统的资源子系统，该子系统使用层次型架构，上层主Agent对下层从Agent具有管理、指导和协调作用。在分布式环境中，上层Agent中记录有下层Agent的IP地址和端口号，同时每个Agent内部都开辟一个线程池，用于处理来自上层和下层Agent的Socket请求。线程池我们选用定长线程池，即限定线程的最大并发数，超出的线程会在队列中等待，这样可以避免当Socket请求过多而导致系统资源的使用超过负载，造成系统运行性能的急剧降低，如下图所示：



上层Agent把任务分解后，将子任务合同封装成Socket消息传递到下层Agent，下层Agent同样把任务进行分解后再发布到再下层子Agent，这样任务的分解和分配就层层向下传递，由下层Agent进行竞标和投标。传递的每一个消息都需要注明是合同亦或是标书，以辅助Agent进行辨明消息时来源于上层Agent还是下层Agent。为了提高并发程度，合理利用系统资源，对于每一个Socket请求，资源Agent都会从线程池中获取一个线程处理请求，这样上层Agent能够从容灵活地处理来自下层Agent的消息。

**4.3.3 管理Agent与监控Agent之间的通信实现**

管理Agent与监控Agent之间的通信同样是基于Socket的。监控Agent与电科院的工作站建立长久连接，持续地从工作站的数据库获取生产线各个工位的实时数据，把数据存进数据库，进行生产日志的记录，此外还需要对数据进行故障分析。若数据分析过程中发现设备故障，将把故障信息使用Socket通信的方式发送到管理Agent。监控Agent除了进行故障分析外，还需要将生产线各个设备的运行状况生成特定格式的数据，这是为了方便资源Agent模拟的设备运行状况与设备实际的运行状况进行同步，控制时间误差。因此我们在管理Agent中设置专门的定时器用于向监控Agent周期性地获取数据，并把返回数据与各资源Agent的数据进行对比，当两者的数据误差过大，需要重新设置时钟和工作进度。

其次，各Agent与工艺Agent和算法Agent的通信方式较为相似，在此不再赘述。

**4.3.4 与现有系统的衔接**

为了降低开发费用，节省开发成本，使实现的调度系统尽快投入使用，因此将桂刘冰师兄的生产线监控系统作为独立的监控Agent。监控系统原有的数据采集和故障分析功能保持不变，增加Socket通信与管理Agent相连接，同时将生产线的实时数据封装成特定的格式，以便于与资源Agent模拟的设备工作状态进行对比，控制两者之间的时间误差。

由于师兄的监控系统是使用MFC实现，而目前的多Agent调度原型系统是基于Java实现的，为了降低系统集成的难度，我们在监控系统中集成了多Agent系统，实现方法是Sun公司的JNI技术。

JNI即Java Native Interface 。从Java 1.1开始，Java Native Interface (JNI)标准成为java平台的一部分，它允许Java代码和其他语言写的代码进行交互。因此我们可以使用JNI在C/C++代码内调用Java类方法，要做到这一点**必须使用 Invocation API 在本机代码内创建和初始化一个 JVM。具体步骤通常分为以下几步：编写、编译Java代码，加载JVM动态库，创建嵌入式JVM，使用特定的API调用Java类的静态方法，或者生成Java的类对象，从而调用类方法。**

**实现过程需要添加JVM动态库的链接，调用**调用JNI\_CreateJavaVM函数创建JVM。该函数会传出一个JNI上下文环境对象（JNIEnv），利用该对象的相关函数就可以调用JAVA类的属性和方法了

**4.4 各类Agent的程序设计**

在第二章中，我们阐述了各个Agent的内部模型，其中的各个部分代表了特定Agent所具有的功能。例如管理Agent具有动态数据库、静态数据库、自学习模块、注册/注销模块、推理机、调度执行模块和通信管理模块。这里，我们主要给出管理Agent和资源Agent的程序设计，如其中的属性。方法、继承的基类和实现的接口等。

管理Agent的类设计如下所示：



资源Agent主要分为三类Agent：SystemAgent、JobShopAgent和MachineAgent，分别代表制造系统、车间以及设备资源。三类Agent的继承关系如下图所示。



下面主要列出SystemAgent的类设计。其中主要涉及任务的分解，即投标—竞标—中标三个过程。



这一章主要阐述如何使用面向对象技术实现基于多Agent的动态调度原型系统的系统架构。在面向对象的Java语言的支持下,定义了Agent的功能接口以及对系统内公共数据库等关键组件进行了对象化的封装,通过对象之间的消息传递,体现系统的社会性、智能型与合作能力。在这种框架下,使得多Agent车间调度系统的开放性、模块化、稳定性和易维护性等特性得以实现。